HANSER

Leseprobe

Peter Armstrong

Flexible Rails

Flex 3 auf Rails 2

Übersetzt aus dem Englischen von Dorothea Heymann-Reder

ISBN: 978-3-446-41573-7

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-41573-7

sowie im Buchhandel.

3 Erste Schritte

Forsan et haec olim meminisse juvabit
(„Vielleicht wird es einst eine Freude sein, auch dieser Dinge zu gedenken.“)

– Vergil, Aen. I, 203

In dieser Iteration machen wir erstmals mit unserer Anwendung Fortschritte. Wir verfügen dann über eine gute Grundlage für jede Flex+Rails-Anwendung, natürlich ganz besonders für die Pomodo-Anwendung. Als Erstes frieren wir die Rails-Version ein, installieren `restful_authentication` und führen es aus. Danach erstellen und starten wir die Migration zur Einrichtung von Benutzern, um als Nächstes wieder zur Flex-Seite zu wechseln und die Benutzer-Erstellung und -Anmeldung zum Laufen zu bringen. Abschließend vergewissern wir uns, dass unsere kleine Test-Suite funktioniert. Durch diese Iteration werden Sie sich durchbeißen müssen, daher auch das vorangestellte Zitat. Doch ich kann Ihnen versichern: Wenn Sie das geschafft haben, wird das Buch um vieles spaßiger.

Hinweis

Die Ordner in der Zip-Datei der Codebeispiele zeigen den jeweiligen Zustand am Ende der Iteration. Mit jeder Iteration von 3–12 können Sie beginnen, indem Sie den Code aus dem Ordner der vorherigen Iteration laden. Wenn Sie die ersten Iterationen nicht nachvollziehen möchten, sondern gleich in Iteration 9 einsteigen, sollten Sie die Iterationen 1–8 immerhin durchlesen und dann die Iteration 9 mit dem Code des Ordners `iteration08` beginnen. Benennen Sie das Projekt von Iteration08 in pomodo um, und importieren Sie es mit dem nachfolgend dargestellten Verfahren.

3.1 Wenn Sie hier zu lesen beginnen

▶ Wenn Sie bis hierher alles gelesen haben, können Sie zu Abschnitt 3.2 übergehen.

Wenn Sie erst hier zu lesen beginnen und ab jetzt das Buch nachvollziehen möchten, kopieren Sie den für Iteration 2 passenden Quellcode-Ordner (iteration02_flexbuilder_windows, iteration02_ sdk_mac oder iteration02_sdk_windows) aus der Code-Zip-Datei, fügen ihn in Ihr Arbeitsverzeichnis ein und benennen dieses in `current` um. Wenn Sie Flex Builder verwenden, bearbeiten Sie die Datei .project mit einem Editor und benennen das Projekt von `iteration02` in `pomodo` um, wie in Listing 3.1 gezeigt.

Listing 3.1 .project-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
      <name>pomodo</name>
      <comment></comment>
...
</projectDescription>
```

Nun wählen Sie File > Import > Other und wählen General / Existing Projects into Workspace (Abbildung 3.1).

Abbildung 3.1
Ein vorhandenes Projekt (keinen Zip-Export) in Flex Builder importieren (Schritt 1)

Hinweis
Wir wählen nicht File > Import > Flex Project, weil dies in FlexBuilder 3 Beta 2 und niedriger voraussetzt, dass eine exportierte Zip-Projektdatei zum Importieren bereitsteht. In Flex Builder 3 Beta 3 (und wohl auch den höheren Versionen) funktioniert es: Hier können Sie einen Projektordner für den Import angeben.

Klicken Sie auf Next, um den Dialog Import Projects aufzurufen (Abbildung 3.2). Nun klicken Sie auf Browse, gehen zum Speicherort des neuen Verzeichnisses `current\`

`pomodo` und klicken auf OK. Das Stammverzeichnis wird eingerichtet und das `pomodo`-Projekt markiert. Wählen Sie jetzt nicht Copy projects into workspace, denn das Projekt soll bleiben, wo es ist. Das Dialogfeld sollte nun aussehen wie in Abbildung 3.2.

Klicken Sie auf Finish.

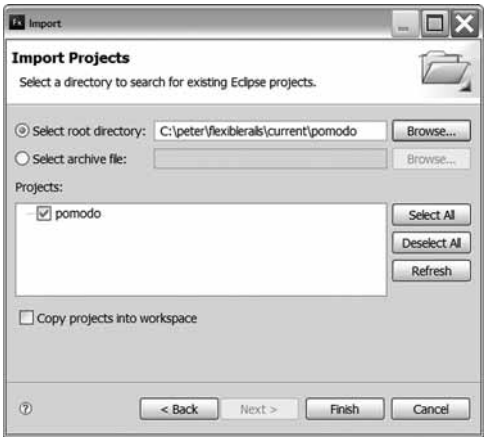

Abbildung 3.2
pomodo in Flex Builder importieren

Frage: Warum muss ich das Projekt in der .project-Datei umbenennen, wenn ich mit einem Projekt aus der Code-Zip-Datei starte, die ich von http://www.flexiblerails.com/code-samples heruntergeladen habe. Sollte das Projekt nicht `pomodo` heißen?

Antwort: Als ich das Buch schrieb, musste ich alle Iterationen pflegen, um die Fehler darin zu beheben (ja, auch ich mache Fehler). Also habe ich am Ende jeder Iteration den aktuellen Ordner in einen Ordner kopiert, der nach der jeweiligen Iteration benannt war (zum Beispiel iteration03). Danach habe ich die Datei .project bearbeitet, um das Projekt von `pomodo` in den Namen der Iteration umzubenennen (abermals so etwas wie iteration03). So konnte ich alle Iterationen in Flex Builder gleichzeitig als separate Projekte öffnen, allerdings um den Preis, dass der Code in der Zip-Datei Projekte mit Namen iteration03, iteration04 anstatt ein Projekt namens pomodo enthielt. Dies ist der Grund, weshalb Sie am Anfang jeder Iteration das Projekt in der .project-Datei umbenennen müssen, wenn Ihr Projekt immer noch `pomodo` heißt. Abschließend sollten Sie wissen, dass ich nur für die Iteration 2 drei Quellcode-Ordner erstellt habe: iteration02_flexbuilder_windows, iteration02_sdk_mac und iteration02_sdk_windows. (Die Ordner der restlichen Iterationen legen die Kombination von Flex Builder und Windows zugrunde.)

Bevor wir beginnen, neue Features hinzuzufügen, müssen wir noch zwei Kleinigkeiten erledigen: die Rails-Version einfrieren und die Integration der Browser-Navigation ausschalten.

3.2 Einfrieren der Rails-Version

Hinweis

Dieser Abschnitt ist optional. Sie können dem Buch auch folgen, wenn Sie ihn überspringen (ausgenommen den Teil, in dem ich Sie den Quellcode in vendor\rails lesen lasse). Wenn Ihnen also dieser Abschnitt zu schwierig ist, gehen Sie einfach zu Abschnitt 3.3. Außerdem setzen die folgenden Ausführungen voraus, dass Sie Subversion installiert haben und auf der Kommandozeile arbeiten. Um dies zu überprüfen, geben Sie an der Eingabeaufforderung `svn-- version` ein: Sie müssten nun eine Versionsangabe von svn sehen, etwa „svn, version 1.4.2“. Hilfe bei der Installation von Subversion finden Sie in Anhang A.

Das Einfrieren der Rails-Version gewährleistet, dass der Rails-Code weiterhin funktionieren wird, wenn Sie die Iterationen 3–12 zu einem späteren Zeitpunkt noch einmal laden möchten (und wenn Rails in der Zwischenzeit deutliche Änderungen durchgemacht hat). Außerdem ist es eine sinnvolle Maßnahme, weil es uns ermöglicht, den Rails-Quellcode zu durchschauen. In Abschnitt 2.1.2 haben Sie entweder die neueste und beste Version von Rails 2 installiert oder den Release Candidate (1.99.0, auch RC1 genannt) von Rails 2, der während der Herstellung dieses Buchs verwendet wurde. In diesem Abschnitt werden Sie eine bestimmte Version von Rails festlegen („einfrieren“). Für die Release Candidate-Version RC1 führen Sie hierzu folgenden Befehl im pomodo-Verzeichnis aus:

```
c:\peter\flexiblerails\current\pomodo>
rake rails:freeze:edge TAG=rel_2-0-0_RC1
...weitere Ausgabe wurde weggelassen...
```

Um den Rails-Release 2.0 einzufrieren, lautet der Befehl wie folgt:

```
c:\peter\flexiblerails\current\pomodo>
rake rails:freeze:edge TAG=rel_2-0-1
...weitere Ausgabe wurde weggelassen...
```

So wird ein rails-Verzeichnis erstellt, das allen Quellcode von Rails im vendor-Verzeichnis enthält. Unsere Anwendung wird in diesem Verzeichnis die Version von Rails 2 verwenden, keine anderweitigen, als Gems installierten Versionen.

Das Einfrieren der Rails-Version empfiehlt sich auch, wenn Sie in einer Umgebung arbeiten, in der Sie keine Kontrolle darüber haben, welche Version von Rails installiert wird (wie zum Beispiel in einer gemeinsam genutzten Hosting-Umgebung). Eigentlich ist es für jede Bereitstellung empfehlenswert. Punkt.

3.3 Integration der Browser-Navigation deaktivieren

Um sicherzustellen, dass Sie den Code in diesem Buch möglichst reibungslos nachvollziehen, müssen wir noch eines tun: die integrierte Browser-Navigation ausschalten . Flex 3 besitzt die Fähigkeit, die Browser-Navigation zu integrieren, damit beispielsweise so etwas wie ein Klick des Anwenders auf den Zurück-Button auch in Ihrer Anwendung eine Wirkung hat. So nett das ist, macht es doch die Arbeit zu einer Qual, weil es ärgerliche Seiten-

effekte haben kann, eine Anwendung neu zu laden, wenn im URL eine Menge zusätzliches Zeugs enthalten ist. Daher werden wir diese Funktion für die Zwecke dieses Buchs ausschalten.

Hinweis

SDK-Benutzer können diesen Abschnitt überspringen, da er nur Flex Builder betrifft. (Sie können dieses Feature verwenden, müssen es aber selbst hinzufügen, statt es auszuschalten.)

Klicken Sie mit der rechten Maustaste auf das pomodo-Projekt im Flex Navigator, und wählen Sie die Option Flex Compiler. Als Nächstes deaktivieren Sie das Kontrollkästchen Enable Integration with Browser Navigation im Abschnitt HTML Wrapper, wie in Abbildung 3.3 gezeigt.

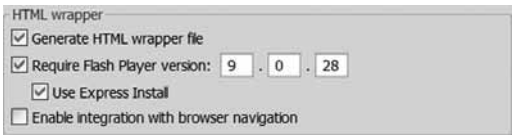

Abbildung 3.3
Integrierte Browser-Navigation im HTML-Wrapper ausschalten

Nachdem Sie auf OK geklickt haben, erscheint die Warnung aus Abbildung 3.4.

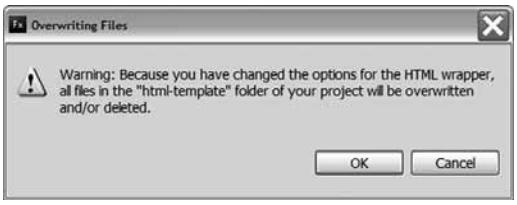

Abbildung 3.4 Warnung beim Überschreiben von Dateien durch HTML-Wrapper

Klicken Sie auf OK.

Da wir nun endlich alles richtig eingerichtet haben, können wir damit beginnen, etwas zu erstellen.

3.4 Benutzeranmeldung in Rails hinzufügen

Da wir eine To-Do-Listenanwendung für mehrere Benutzer im GTD-Stil erstellen möchten, müssen wir irgendwann auch eine Benutzeranmeldung (Login-Funktionalität) implementieren. Tricksen wir also ein wenig: Weiter unten, in Iteration 5, stellen wir den Rails-Code durch Refactoring auf RESTful um, und da wir dies nun wissen, können wir von Anfang an die RESTful-Authentifizierungsmethode verwenden, ohne dass ich den REST erklären muss. Also werde ich, wenn die Erklärung von REST an der Reihe ist, auch den Login-Code erläutern, den wir bereits benutzt haben, und dann den Rails-Code auf RESTful Authentication umstellen. Würden wir einen anderen Login-Generator verwenden, müssten wir dessen Ausgabe entweder umstellen oder wegwerfen und dann dasselbe tun, was wir jetzt tun werden.

Rails Engines und Wiederverwendbarkeit

Die Unfähigkeit zur Wiederverwendung der Generator-Ausgabe ist einer der Gründe für Rails Engines (http://rails-engines.org/). Leider besteht in der Rails-Community eine Kontroverse bezüglich Rails Engines, die wir in diesem Buch nicht lösen können. Es wäre jedoch unfair, über Wiederverwendung zu reden, ohne Rails Engines zu erwähnen. Mein Rat lautet: Sobald Sie genug über Rails wissen, um einen Entwurf in Rails zu verstehen, gehen Sie auf die Homepage von Rails Engines, und bilden Sie sich eine eigene Meinung.

Wir werden nicht nur den Login-Generator restful_authentication installieren, sondern auch unseren Rails-Code so bearbeiten, dass der generierte Code mit den HTML-Views funktioniert. Zwar handelt das Buch im Wesentlichen von Flex mit Rails, aber auch HTML hat immer noch seinen Platz, wenn es um Scaffolding und Ad-hoc-Testen von Controllern geht.

3.4.1 Installation und Ausführung von *restful_authentication*

Wir beginnen, indem wir das restful_authentication-Plugin von Rick Olson installieren.

Tipp

Dieses Plugin wird auch von Geoffrey Grosenbach („Mr. Topfunky„) in seinem Screencast „PeepCode RESTful Rails“ verwendet (http://peepcode.com/articles/2006/10/08/restful-rails). Ich hielt diesen Screencast für eine nützliche Ergänzung zu den Diskussionen über REST in Agile Web Development with Rails (AWDwR) und in „Discovering a world of Resources on Rails“, einer Keynote zur RailsConf 2006 von DHH (http://media.rubyonrails.org/presentations/worldofresources.pdf). Der Screencast „PeepCode RESTful Rails“ ist eineinhalb Stunden lang, aber unterhaltsamer als so mancher Spielfilm. Ich empfehle, ihn zu kaufen und vor dem Lesen von Iteration 5 anzuschauen.

Wir installieren nun das Plugin restful_authentication.[1] Führen Sie folgenden Befehl aus, und achten Sie darauf, welche Dateien er anlegt (und lesen Sie die Usage-Meldung):

```
c:\peter\flexiblerails\current\pomodo>
ruby script\plugin install -r 3072 http://svn.techno-
     ➥weenie.net/projects/plugins/restful_authentication/
+ ./README
+ ./Rakefile
+ ./generators/authenticated/USAGE
...Ausgabe wurde weggelassen...
c:\peter\flexiblerails\current\pomodo>
```

Lesen Sie unbedingt die Usage-Meldung, die der Installer auf der Kommandozeile ausgibt. Dieser Befehl installiert die Version 2007-12-12 von restful_authentication im Verzeichnis pomodo\vendor\plugins\restful_authentication. (Wir verwenden eine bestimmte Version, um sicherzustellen, dass der generierte Code mit dem Buch übereinstimmt.)

[1] http://peepcode.com/articles/2006/10/08/restful-rails um 04:54 im Screencast (peepcode-003-rest.mov)

Jetzt wollen wir den restful_authentication-Generator auch benutzen. Wir geben nicht die Option `--include-activation` an, weil eine Bestätigung der Benutzerkonto-Eröffnung per E-Mail ebenso wie alles andere aus `ActionMailer` nicht Thema dieses Buches ist. Da ich Windows zugrunde lege, führe ich die Skripte mit `ruby` statt `./` aus:

Hinweis

Benutzer von Mac OS X und Linux müssen wissen, dass das Buch ab jetzt Windows voraussetzt, denn irgend etwas musste ich schließlich zugrunde legen. Sie müssen den normalen Schrägstrich / anstelle des Backslashs \ benutzen und auf OS X, wenn ein Befehl nicht funktioniert, `sudo` davor setzen. (Wenn Sie Flex Builder benutzen, ändert sich in Wirklichkeit nicht viel.)

```
c:\peter\flexiblerails\current\pomodo>
ruby script\generate authenticated user sessions
...Ausgabe wurde weggelassen...
```

Die Ausführung dieses Generators erzeugt eine Menge neuer Dateien und Verzeichnisse sowie eine Migration[2] (db\migrate\001_create_users.rb). Es ist immer sinnvoll, den gesamten Code, den ein Generator erstellt, durchzulesen.

3.4.2 CreateUsers-Migration bearbeiten, ausführen und Resultate prüfen

Als Nächstes bearbeiten wir die Migration, um die Vor- und Nachnamen hinzuzufügen (Listing 3.2).

Listing 3.2 db\migrate\001_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table "users", :force => true do |t|
      t.column :login,                     :string
      t.column :email,                     :string
      t.column :first_name,                :string, :limit => 80   (1)
      t.column :last_name,                 :string, :limit => 80   (2)
      t.column :crypted_password,          :string, :limit => 40
      t.column :salt,                      :string, :limit => 40
      t.column :created_at,                :datetime
      t.column :updated_at,                :datetime
      t.column :remember_token,            :string
      t.column :remember_token_expires_at, :datetime
    end
  end

  def self.down
    drop_table "users"
  end
end
```

[2] Migrationen werden im folgenden Blogbeitrag sehr gut erklärt: http://glu.ttono.us/articles/2005/10/27/the-joy-of-migrations.

Der restful_authentication-Generator erstellt von `ActiveRecord::Migration` eine Unterklasse namens `CreateUsers`, deren `up`-Methode die Benutzertabelle `users` in einer datenbankunabhängigen Weise erstellt und deren `down`-Methode diese Tabelle ebenso datenbankunabhängig löscht. Den gesamten Code fügt der Generator selbsttätig hinzu; wir müssen nur noch Spalten für die Vornamen **(1)** und Nachnamen **(2)** anlegen. In beiden Fällen legen wir 80 Zeichen als maximale Länge fest.

Nun gehen Sie wieder zur Eingabeaufforderung Ihres WEBrick-Servers und halten diesen mit Strg-C an. Da wir die Datenbankkonfiguration in `config\database.yml` geändert haben, müssen wir den Server neu starten, beschränken uns hier jedoch darauf, ihn herunterzufahren; das Hochfahren erledigen wir später.

Weil wir nun die Migration erstellt haben, die unsere Benutzertabelle erzeugen wird, wollen wir das Skript newdb.bat laufen lassen, um die Datenbanken neu zu erstellen und die Migration auszuführen:

```
c:\peter\flexiblerails\current\pomodo>newdb.bat

c:\peter\flexiblerails\current\pomodo>
mysql -h localhost -u root -p  0<db\create.sql
Enter password: *******

c:\peter\flexiblerails\current\pomodo>call rake db:migrate
c:0:Warning: require_gem is obsolete.  Use gem instead.
(in c:/peter/flexiblerails/current/pomodo)
== 1 CreateUsers: migrating ===================================================
-- create_table("users", {:force=>true})
   -> 0.2500s
== 1 CreateUsers: migrated (0.2500s) ==========================================


c:\peter\flexiblerails\current\pomodo>
```

Überzeugen Sie sich, dass Datenbanken und users-Tabelle angelegt wurden:

```
c:\peter\flexiblerails\current\pomodo>mysql -u root -p
Enter password: *******
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 6 to server version: 5.0.24-community-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| pomodo_development |
| pomodo_production  |
| pomodo_test        |
| test               |
+--------------------+
6 rows in set (0.00 sec)

mysql>
```

So weit, so gut. Mal sehen, wie die pomodo_development-Datenbank-Database aussieht:

```
mysql> use pomodo_development;
Database changed
mysql> show tables;
+------------------------------+
| Tables_in_pomodo_development |
+------------------------------+
| schema_info                  |
| users                        |
+------------------------------+
2 rows in set (0.00 sec)

mysql> describe users;
+---------------------------+--------------+------+-----+---------+
| Field                     | Type         | Null | Key | Default |
+---------------------------+--------------+------+-----+---------+
| id                        | int(11)      | NO   | PRI | NULL    |
| login                     | varchar(255) | YES  |     | NULL    |
| email                     | varchar(255) | YES  |     | NULL    |
| first_name                | varchar(80)  | YES  |     | NULL    |
| last_name                 | varchar(80)  | YES  |     | NULL    |
| crypted_password          | varchar(40)  | YES  |     | NULL    |
| salt                      | varchar(40)  | YES  |     | NULL    |
| created_at                | datetime     | YES  |     | NULL    |
| updated_at                | datetime     | YES  |     | NULL    |
| remember_token            | varchar(255) | YES  |     | NULL    |
| remember_token_expires_at | datetime     | YES  |     | NULL    |
+---------------------------+--------------+------+-----+---------+
11 rows in set (0.01 sec)

mysql> select * from users;
Empty set (0.00 sec)
```

Sieht gut aus. Beachten Sie, dass die Spalte „Extra“ hier aus Platzgründen nicht gezeigt wird, da sie lediglich anzeigt, dass die id-Spalte auto_increment ist. Beachten Sie außerdem die Felder first_name und last_name, die wir hinzugefügt haben, und das neue id-Feld, das den Primärschlüssel darstellt. Es wurde automatisch erzeugt, obwohl wir es in unserer Migration gar nicht angegeben hatten, denn ein Grundprinzip von Rails lautet: Konvention geht vor Konfiguration – und eine seiner Konventionen ist nun einmal, dass ein Primärschlüssel namens id vorhanden sein muss.

Doch was für eine Tabelle ist schema_info?

Die kurze Antwort lautet, dass sie durch das Ausführen der Migration angelegt wurde und über ein einziges Feld namens version verfügt, das die Version des Datenbankschemas speichert:

```
mysql> describe schema_info;
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| version | int(11) | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
1 row in set (0.01 sec)

mysql> select * from schema_info;
+---------+
| version |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)

mysql>
```

Momentan gilt die Version 1, sodass das Migrationsskript weiß, welche Version des Datenbankschemas gerade aktuell ist, wenn Sie auf eine andere Version umstellen.

Bevor wir weitermachen, sollten wir das User-Modell bearbeiten, um die neuen Vor- und Nachnamenattribute (`first_name` und `last_name`) aus dem `params`-Hash en bloc zuweisen zu können. (Wenn das zurzeit noch unverständlich ist, machen Sie sich keine Sorgen – wir kommen später darauf zurück.) Bearbeiten Sie das `User`-Modell, wie in Listing 3.3 gezeigt.

Listing 3.3 app\models\user.rb

```
...
  # hindert den Benutzer am Senden eines selbsterstellten Formulars,
  # das die Aktivierung umgeht.
  # Auch alles andere, was der Benutzer ändern können soll,
  # muss hier hinzugefügt werden.
  attr_accessible :login, :email, :password,
    :password_confirmation, :first_name, :last_name  ‖ (1)
...
```

Wir fügen die neuen Attribute `first_name` und `last_name` **(1)** zu demselben `attr_accessible`-Aufruf hinzu wie alle anderen Attribute. (Die Attributnamen fangen mit einem Doppelpunkt an, weil sie zur Klasse `Symbol` gehören; auch darüber später mehr.)

Als Nächstes fügen wir zu config\routes.rb die RESTful-Routen hinzu, wie es der Generator von uns verlangt.

3.4.3 RESTful-Routen hinzufügen

Kopieren Sie den Code aus den Anweisungen, die der Generator auf der Kommandozeile ausgegeben hat, und fügen Sie ihn ein, wie in Listing 3.4 gezeigt.

Listing 3.4 config\routes.rb

```
ActionController::Routing::Routes.draw do |map|
  # Priorität richtet sich nach der Reihenfolge der Erstellung:
  # zuerst erstellt -> höchste Priorität.
...
  # Beispiel für Ressourcen-Route(bildet automatisch HTTP-Verben
  # auf Controller-Aktionen ab):
  #   map.resources :products
  map.resources :users     ‖ (1) User-Ressource
  map.resource  :session   ‖ (2) Session-Ressource

  map.signup '/signup', :controller => 'users',   ‖ (3) Anmelde-URL
    :action => 'new'
  map.login  '/login', :controller => 'sessions',  ‖ (4) Anmelde-URL
    :action => 'new'
  map.logout '/logout', :controller => 'sessions',  ‖ (5) Abmelde-URL
    :action => 'destroy'

  # Beispiel für Ressourcen-Route mit Optionen:
  #   map.resources :products,
  #     :member => { :short => :get, :toggle => :post },
  #     :collection => { :sold => :get }
...
end
```

Kurz gefasst kann man dies wie folgt erklären: Wir richten RESTful-Ressourcen für `users` **(1)** und die `session` **(2)** ein und fügen einige spezielle Routen hinzu, um hübschere URLs für `signup` **(3)**, `login` **(4)** und `logout` **(5)** zu erzeugen. Routing ist eine komplizierte Angelegenheit, die man nicht mal so nebenbei in Kapitel 3 eines Buchs erlernt (zumindest nicht in diesem).

Speichern Sie die Datei – so viel zu den Routen. Beim Durchsehen der Dateien, die der Generator für uns erstellt hat (diese müssen Sie immer überprüfen!), haben wir in `UsersController` und `SessionsController` einige Anweisungen bemerkt, die wir nun befolgen werden.

3.4.4 Includes und before_filter anweisungsgemäß modifizieren

Als Erstes bearbeiten wir `SessionsController`, wie in Listing 3.5 gezeigt.

Listing 3.5 app\controllers\sessions_controller.rb

```
# Controller für die An/Abmeldefunktion der Site.
class SessionsController < ApplicationController
  # Be sure to include AuthenticationSystem in
  # Application Controller instead
  include AuthenticatedSystem   ‖ (1) Gelöschtes Include

  # render new.rhtml
  def new
  end
...unveränderter Code wurde weggelassen...
end
```

Wir haben die Include-Anweisung für **(1)** `AuthenticatedSystem` gelöscht.

Als Nächstes bearbeiten wir `UsersController` wie in Listing 3.6.

Listing 3.6 app\controllers\users_controller.rb

```
class UsersController < ApplicationController
  # Be sure to include AuthenticationSystem in Application
  # Controller instead
  include AuthenticatedSystem   ‖ (1) Gelöschtes Include

  # render new.rhtml
  def new
  end

  def create
    @user = User.new(params[:user])
    @user.save!
    self.current_user = @user
    redirect_back_or_default('/')
    flash[:notice] = "Thanks for signing up!"
  rescue ActiveRecord::RecordInvalid
    render :action => 'new'
  end

end
```

Auch hier haben wir **(1)** die Include-Anweisung für `AuthenticatedSystem` gelöscht.

Nun fügen wir die Include-Anweisung, die wir gerade gelöscht haben, in `Application-Controller` ein, wie in Listing 3.7.

Listing 3.7 app\controllers\application.rb

```
# Filter und Methoden in diesem Controller gelten für
# alle Controller der Anwendung

class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  include AuthenticatedSystem  ‖ (1)

  # Details siehe ActionController::RequestForgeryProtection
  # Entkommentieren Sie :secret, wenn Sie den Cookie-Session-
  # Speicher nicht brauchen.
  # TODO - wird entkommentiert, wenn wir Sessions
  # in Iteration 5 erklären.
  # protect_from_forgery  ‖ (2)
  # :secret => 'dd92c128b5358a710545b5e755694d57'
end
```

Indem wir hier die Include-Anweisung für `AuthenticatedSystem` einfügen **(1),** stellen wir sicher, dass alle Controller dieses standardmäßig einbinden (da sie alle die Klasse `ApplicationController` erweitern). Darüber hinaus wird der Aufruf `protect_from_forgery` vorübergehend bis zur Iteration 5 auskommentiert **(2)**, da ich über CSRF-Attacken erst sprechen möchte, wenn klar ist, was eine Session ist. Ihr :secret sollte natürlich ein anderes als dieses hier sein, sonst ist es kein sonderlich geheimes :secret.

Im Weiteren müssen wir das tun, was uns der Testcode (und Geoffrey Grosenbachs Screencast)[3] sagt, indem wir die Zeile `include AuthenticatedTestHelper` zu test\test_helper.rb hinzufügen (siehe Listing 3.8 **(1)**) und aus `UserTest` (Listing 3.9 **(2)**), `Sessions-ControllerTest` (Listing 3.10 **(3)**) und `UsersControllerTest` (Listing 3.11 **(4)**) entfernen.

Listing 3.8 test\test_helper.rb

```
ENV["RAILS_ENV"] = "test"
require File.expand_path(File.dirname(__FILE__) +
  "/../config/environment")
require 'test_help'

class Test::Unit::TestCase
  include AuthenticatedTestHelper  ‖ (1)
...
```

Listing 3.9 test\unit\user_test.rb

```
require File.dirname(__FILE__) + '/../test_helper'

class UserTest < Test::Unit::TestCase
  # Be sure to include AuthenticatedTestHelper in
  # test/test_helper.rb instead.
  # Then, you can remove it from this and the functional test.
  include AuthenticatedTestHelper  ‖ (2)
```

[3] http://peepcode.com/articles/2006/10/08/restful-rails, um 07:05 im Screencast (peepcode-003-rest.mov).

```
  fixtures :users

  def test_should_create_user
...
```

Listing 3.10 test\functional\sessions_controller_test.rb

```
require File.dirname(__FILE__) + '/../test_helper'
require 'sessions_controller'

# Vom Controller gemeldete Fehler erneut auslösen.
class SessionsController; def rescue_action(e) raise e end; end

class SessionsControllerTest < Test::Unit::TestCase
  # Be sure to include AuthenticatedTestHelper in
  # test/test_helper.rb instead
  # Then, you can remove it from this and the units test.
  include AuthenticatedTestHelper  ‖ (3)

  fixtures :users
...
```

Listing 3.11 est\functional\users_controller_test.rb

```
require File.dirname(__FILE__) + '/../test_helper'
require 'users_controller'

# Vom Controller gemeldete Fehler erneut auslösen.
class UsersController; def rescue_action(e) raise e end; end

class UsersControllerTest < Test::Unit::TestCase
  # Be sure to include AuthenticatedTestHelper in
  # test/test_helper.rb instead
  # Then, you can remove it from this and the units test.
  include AuthenticatedTestHelper  ‖ (4)

  fixtures :users
...
```

Ich werde in diesem Buch nicht viel mehr übers Testen sagen. Allerdings müssen wir dafür sorgen, dass unsere Tests am Ende dieser Iteration alle gelingen, damit wir zumindest so tun können, als führten wir richtige Tests durch .

Hinweis

Testen ist sehr wichtig, und der Umstand, dass ich das gesamte Thema auf einen kleinen Abschnitt in Anhang B verlagere, soll keinesfalls meine Meinung über die Bedeutung von Tests widerspiegeln, sondern vielmehr meinen Wunsch, dieses Buch solle Lesevergnügen bereiten. Es handelt sich eben um kein Buch über Tests.

So viel zu den Modifikationen des generierten Codes. Als Nächstes testen wir, wie wir Benutzerkonten mit den HTML-Views anlegen.

3.4.5 Anlegen von Benutzerkonten in HTML testen

Auch wenn unser Buch von der Kombination von Flex und Rails lebt, ist es gelegentlich von Vorteil, Rails in HTML zu testen (und zwar besonders mit Tools wie Firebug, um Formularwerte entgegennehmen zu können usw.). Wir wollen uns vergewissern, dass das Anlegen von Benutzerkonten und das Anmelden funktioniert.

Hinweis

Wenn Sie Ihren Server schon seit Iteration 2 laufen lassen, halten Sie ihn nun mit Strg-C an. Da wir Konfigurationsdateien geändert haben, müssen wir den Server neu starten.

Starten Sie den Server:

```
c:\peter\flexiblerails\current\pomodo>ruby script\server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
```

Öffnen Sie nun ein Browserfenster, und gehen Sie zu http://localhost:3000/signup. Wir haben in config\routes.rb die folgende Route definiert:

```
  map.signup '/signup', :controller => 'users',
    :action => 'new'
```

Dies wird die neue Aktion des `UsersController` in Gang setzen (siehe Listing 3.12).

Listing 3.12 app\controllers\users_controller.rb

```
class UsersController < ApplicationController
  # render new.rhtml     ‖ (1)
  def new
  end
...
```

Da diese Aktion leer ist, wird entsprechend dem Kommentar **(1)** die Datei new.rhtml angezeigt (siehe Listing 3.13).

Listing 3.13 app\views\users\new.rhtml

```
<%= error_messages_for :user %>
<% form_for :user, :url => users_path do |f| -%>
<p><label for="login">Login</label><br/>
<%= f.text_field :login %></p>

<p><label for="email">Email</label><br/>
<%= f.text_field :email %></p>

<p><label for="password">Password</label><br/>
<%= f.password_field :password %></p>

<p><label for="password_confirmation">Confirm Password</label><br/>
<%= f.password_field :password_confirmation %></p>

<p><%= submit_tag 'Sign up' %></p>
<% end -%>
```

Den Bildschirm sehen Sie in Abbildung 3.5.

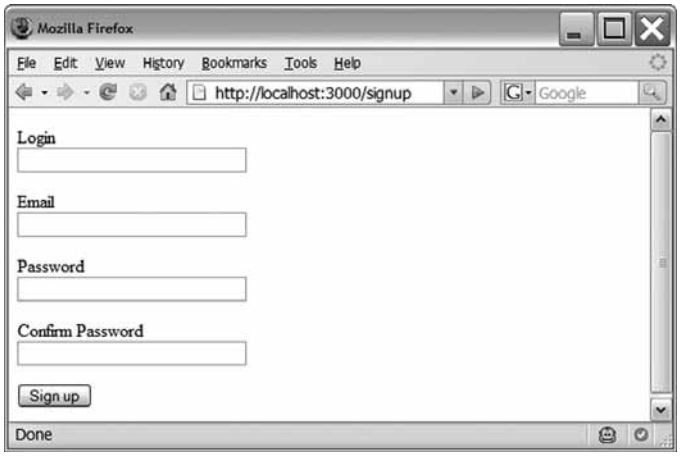

Abbildung 3.5
HTML-Anmeldebildschirm

Geben Sie `ludwig` als Anmeldenamen, `lvb@pomodo.com` als E-Mail-Adresse und `foooo` als Passwort und Passwort-Bestätigung ein.

Klicken Sie nun auf Sign Up, um auf den Rails-Bildschirm Welcome Aboard zu gelangen. Das scheint seltsam, aber da wir keine Standardroute definiert haben, wird die Datei index.html im öffentlichen Verzeichnis angezeigt (eben der Bildschirm Rails Welcome Aboard).

Mal schauen, ob unser Benutzer angelegt worden ist:

```
mysql> select id, login, email, first_name, last_name from users;
+----+--------+----------------+------------+-----------+
| id | login  | email          | first_name | last_name |
+----+--------+----------------+------------+-----------+
|  1 | ludwig | lvb@pomodo.com | NULL       | NULL      |
+----+--------+----------------+------------+-----------+
1 row in set (0.00 sec)

mysql>
```

Prima!

3.4.6 Anmeldung in HTML testen

Versuchen wir, uns anzumelden. Wenn Sie zu http://localhost:3000/login gehen, sehen Sie den Bildschirm aus Abbildung 3.6.

Abbildung 3.6
HTML-Anmeldebildschirm

Als Nächstes wollen wir testen, ob ein fehlerhafter Anmeldeversuch auch pflichtgemäß scheitert. Wenn wir falsche Daten eingeben (entweder das verkehrte Passwort für ludwig oder einen nicht vorhandenen Benutzernamen) und auf Log In klicken, wird wieder der Anmeldebildschirm gezeigt, aber der URL wechselt zu http://localhost:3000/session. Die Gründe dafür werde ich später erklären (schauen Sie in die `create`-Methode von `SessionsController` hinein, wenn Sie neugierig sind). Abbildung 3.7 zeigt den Vorgang.

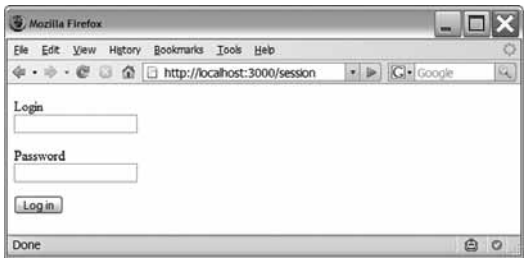

Abbildung 3.7 Fehlgeschlagene HTML-Anmeldung

Wenn wir uns mit dem richtigen Passwort als ludwig anmelden, gelangen wir wieder auf den Bildschirm Welcome Aboard. Das sieht so vertrauenerweckend aus, dass wir nun versuchen werden, uns in Flex anzumelden.

3.5 Benutzeranmeldung in Flex hinzufügen

In diesem Abschnitt statten wir Flex mit einer Funktionalität zur Benutzeranmeldung aus. Wir wollen aber nichts überstürzen, sondern zuerst einmal genau verstehen, was in unserem früheren „Hello World“-Beispiel eigentlich genau geschehen ist.

3.5.1 „Hello World“, diesmal mit Bedeutung!

Kommen wir noch einmal auf Pomodo.mxml zurück, um richtig zu verstehen, was in dem Code eigentlich passiert. Wenn die Anwendung läuft, sieht sie zurzeit aus wie in Abbildung 3.8.

Abbildung 3.8 Pomodo, die Zweite

Den Code in seinem gegenwärtigen Zustand sehen Sie in Listing 3.14.

Listing 3.14 app\flex\Pomodo.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application  ‖ (1)
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"  ‖ (2)
    backgroundGradientColors="[#ffffff, #c0c0c0]"  ‖ (3)
    width="100%"  ‖ (4)
    height="100%">  ‖ (5)
    <mx:HTTPService  ‖ (6)
        id="helloSvc"  ‖ (7)
        url="/hello/sayhello"  ‖ (8)
        method="POST"/>  ‖ (9)
    <mx:Button label="call hello service"
        click="helloSvc.send()"/>  ‖ (10)
    <mx:TextInput text="{helloSvc.lastResult}"/>  ‖ (11)
</mx:Application>
```

Wir wollen verstehen, was hier passiert. Betrachten Sie als Erstes das Tag ganz oben: `mx:Application` **(1)**. Die Wurzel einer Flex-Anwendung ist immer eine `Application`. Das `mx:` gibt an, aus welchem XML-Namensraum[4] die `Application`-Komponente stammt. Wir verwenden ein vertical als Layout-Angabe **(2)**, um die Komponenten vertikal fließen zu lassen. Andere Möglichkeiten sind horizontal (für horizontales Fließen) und absolute (wobei die x,y-Koordinaten für jeden übergeordneten Container angegeben werden müssen). Die `backgroundGradientColors` **(3)** geben an, wo der Farbverlauf für die Hintergrundfarben anfängt und aufhört – und welche Anwendung der Web 2.0-Ära könnte darauf schon verzichten?

Der `HTTPService` **(6)** ruft die `sayhello`-Methode des `HelloController` auf, da dies die Art ist, wie Rails Routing bewerkstelligt. Vereinfacht ausgedrückt, werden beim Standard-Routing URLs zu `/controller_name/action_name` zugeordnet. Dabei wird der Name des Controllers groß geschrieben und der Controller-Teil als gegeben vorausgesetzt, sodass /hello auf `HelloController` und sayhello auf die Aktion `sayhello` (eine Methode) abgebildet wird, während `url="/hello/sayhello"` **(8)** der `sayhello`-Methode der Klasse `HelloController` zuzuordnen ist. Im Moment übergeben wir noch keine Parameter; dies werden Sie schon bald tun, wenn Sie sich anmelden.

Die Kennung (ID) von `HTTPService` ist `helloSvc` **(7)**. In MXML wird die `id`-Property einer Komponente zu ihrem Variablennamen (die MXML-Datei ist eine Klasse und die ID der Name einer globalen Variablen innerhalb dieser Klasse). Wenn wir keine ID für eine Komponente bereitstellen, denkt sich Flex eine für uns aus – aber dann wissen wir nicht, wie sie lautet, und können uns in unserem Code folglich nicht auf sie beziehen. Manchmal ist das auch nicht nötig: Da wir den `TextInput` nicht referenzieren müssen, können wir uns die Mühe sparen, ihm eine ID zu geben. Der `helloSvc HTTPService` benötigt hingegen sehr wohl eine ID, damit wir seine `send`-Methode aufrufen können, wenn das `click`-Event des Buttons eintritt **(10)**, und damit wir uns in der Bindung der `text`-Property von

[4] `<foo:Application xmlns:foo="http://www.adobe.com/2006/mxml" ...` wäre ebenfalls möglich gewesen, aber das würde jeden Flex-Programmierer nur verwirren, weil `mx:` nun einmal die Konvention ist.

`TextInput` **(11)** auf das letzte Ergebnis dieses `HTTPService` beziehen können. Beachten Sie, dass wir die HTTP-`POST`-Methode verwenden müssen **(9)**, da die `method`-Property von `HTTPService` den Standardwert `GET` hat. In Iteration 8 werden wir noch sehr viel mehr über HTTP sprechen.

Abschließend geben die Breite (`4`) und Höhe (`5`) von 100% an, wie groß die Flex-Anwendung im Vergleich zur Größe des Browserfensters sein soll. Wir können Absolutwerte in Pixeln angeben, so sieht zum Beispiel `width="300" height="100"` aus wie in Abbildung 3.9 (beachten Sie den weißen Hintergrund der Webseite, in der die Flex-Anwendung läuft).

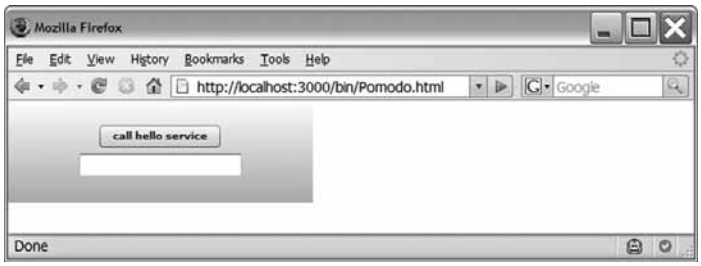

Abbildung 3.9
Die Größe einer Flex-Anwendung mit Absolutwerten für Breite und Höhe einstellen (`width="300" height="100"`)

3.5.2 Bindung? Was ist das nun schon wieder…?

Bindung ist ein kompliziertes Konzept, das sich grob vereinfacht in einem Satz zusammenfassen lässt: „Es kopiert automatisch den Wert der Variablen `x` in die Variable `y`, wann immer sich `x` ändert." Um jedoch voll und ganz zu verstehen, was Bindung ist, muss viel weiter ausgeholt werden. Im Flex 3 Developer's Guide in der Zip-Datei mit der Flex 3 Beta 2-Dokumentation wird das Thema vollständig behandelt, das wir hier nur zusammenfassen können. Die geschweiften Klammern in `text="{helloSvc.lastResult}"` (**(11)** in Listing 3.14) weisen auf eine Bindung hin, das heißt, der Wert von `helloSvc.lastResult` sollte in `resultTI.text` kopiert werden, wann immer er sich ändert.

Hinweis

Nicht alle Variablen können als Quelle für eine Bindung verwendet werden: Hierfür eignen sich nur `Bindable`-Variablen. Dies wurde in Flex 2 neu eingeführt (und gilt immer noch in Flex 3), während Sie in Flex 1.5 und älter eigentlich alles als Quelle für eine Bindung benutzen konnten, wobei aber manche Dinge nicht funktionierten (was sehr vereinfacht dargestellt ist, aber glauben Sie mir: Es ist besser, Sie sparen sich die Einzelheiten).

Jedenfalls ist Bindung sowohl mit einem `<mx:Binding>`-Element als auch mit der kürzeren Syntax der geschweiften Klammern möglich. Der Code in Listing 3.15 ist äquivalent zur Original-Pomodo.mxml-Datei. (Nehmen Sie aber jetzt nicht diese Änderung vor, oder machen Sie sie wieder rückgängig, wenn Sie es getestet haben.)

Listing 3.15 app\flex\Pomodo.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    backgroundGradientColors="[#ffffff, #c0c0c0]"
    width="100%"
    height="100%">
    <mx:HTTPService
        id="helloSvc"
        url="/hello/sayhello"
        method="POST"/>
    <mx:Button label="call hello service"
        click="helloSvc.send()"/>
    <mx:Button label="call hello service" click="helloSvc.send()"/>
    <mx:TextInput id="resultTI"/>  ‖ (1)
    <mx:Binding source="String(helloSvc.lastResult)"
        destination="resultTI.text"/>  ‖ (2)
</mx:Application>
```

Beachten Sie, dass wir dem `TextInput` **(1)** eine ID geben mussten, um ihn in der `destination`-Property der Bindung ansprechen zu können **(2)**. Außerdem mussten wir die Quelle in den Typ `String` umwandeln, was bei Verwendung der geschweiften Klammern nicht notwendig ist.

3.5.3 Dieses MXML sieht seltsam aus

Es stimmt, dass MXML seltsam anmutet, wenn man es zum ersten Mal sieht – selbst wenn XML bereits bekannt ist. Ein Attribut einzustellen, kann hier mehrerlei bedeuten. Betrachten Sie zum Beispiel den folgenden `Button` (dies ist gültiges MXML, wenn die anderen Funktionen und Variablen vorhanden sind):

```
<mx:Button id="bar" label="foo"
  enabled="{someFunctionThatReturnsBoolean(someArg, someOtherArg)}"
  color="#CCDDEE" click="doSomething()"/>
```

Hier geschieht eine ganze Menge:

- Der `Button` wird mit `bar` als `id` angelegt, anstatt einer automatisch zugewiesenen `id`.
- Die `label`-Property des `Button` wird auf `"foo"` gesetzt.
- Die `enabled`-Property von `Button` wird an das Ergebnis der Funktion `someFunctionThatReturnsBoolean` gebunden, die automatisch jedesmal aufgerufen wird, wenn sich `someArg` oder `someOtherArg` ändert. (Das setzt voraus, dass `someArg` und `someOtherArg` `Bindable` sind.)
- Der `color`-Style des `Button` wird auf `#CCDDEE` gesetzt.
- Es wird ein Event-Handler für das `click`-Event erzeugt, der bei Eintritt dieses Events die `doSomething()`-Methode aufruft.

Eine einfache Attributzuweisung in MXML kann eine statische (z.B. ein `String`) oder dynamische (z.B. eine `Binding`) Property-Einstellung, eine Style-Einstellung oder eine Event-Handler-Zuweisung sein. Das Erstaunliche daran ist, dass man sich so rasch daran gewöhnt. (Ich weiß nicht, ob es an Flex, dem menschlichen Verstand im Allgemeinen oder

dem Verstand von Programmierern im Besonderen liegt, aber glauben Sie mir: Es wird Ihnen zur zweiten Natur werden.) Außerdem sind die Dokumentationen zur Flex-API gut geeignet, um die Dinge zu identifizieren (Properties, Styles usw. werden darin zusammengefasst).

3.5.4 Flex 3-Dokumentation? Wo denn?

Die Zip-Datei der Flex 3 Beta 2-Dokumentation kann gegenwärtig von http://download.macromedia.com/pub/labs/air/air_b2_docs_flex_100107.zip heruntergeladen werden. Wenn Flex 3 seine Beta-Phase überstanden hat, wird die Dokumentation wahrscheinlich in http://www.adobe.com/support/documentation/en/flex/ verschoben. Zurzeit (im November 2007) befindet sich unter diesem URL immer noch die Flex 2-Dokumentation, da diese der aktuellste Produktions-Release von Flex ist.

Wenn Sie die Datei herunterladen und dekomprimieren, finden Sie darin eine Fülle von erstklassig formulierten Informationen über Flex, und zwar sowohl (natürlich) in Form von PDF-Manuals als auch in Form von API-Docs ähnlich den Java API-Docs.

Die PDFs aus Tabelle 3.1 sind in der Dokumentationsdatei enthalten.

Tabelle 3.1 Flex-Dokumentation

Dateiname	Seitenzahl	Dokumentname
flex3_buildanddeploy.pdf	260	Building and Deploying Flex Applications
flex3_createextendcomponents.pdf	177	Creating and Extending Flex Components
flex3_devguide.pdf	1435	Flex 3 Developer's Guide
flex3_usingflexbuilder.pdf	219	Using Flex Builder
programmingas3.pdf	576	Programming ActionScript 3.0
air_devappsflex.pdf	304	Developing AIR Applications with Adobe Flex

Macht zusammen 2 971 Seiten! Zum Glück lässt sich die Dokumentation in Flex Builder 3 mit Help > Help Contents auch durchsuchen.

Die API-Docs liegen im Ordner langref. Öffnen Sie die Datei index.html, und setzen Sie eine Bookmark darauf, oder stellen Sie sie als Startseite ein. (Ich finde es hilfreich, dass sich diese und die Rails API-Docs automatisch beim Starten von Firefox in Tabs öffnen.)

Abbildung 3.10 zeigt die API-Docs, damit Sie wissen, dass Sie dasselbe wie ich meinen.

Mit dem, was Sie nun wissen, wollen wir versuchen, Konten und Anmeldung von Flex aus zu erstellen. Wir beginnen mit einer Dummy-Benutzeroberfläche (UI). Das wird interessant.

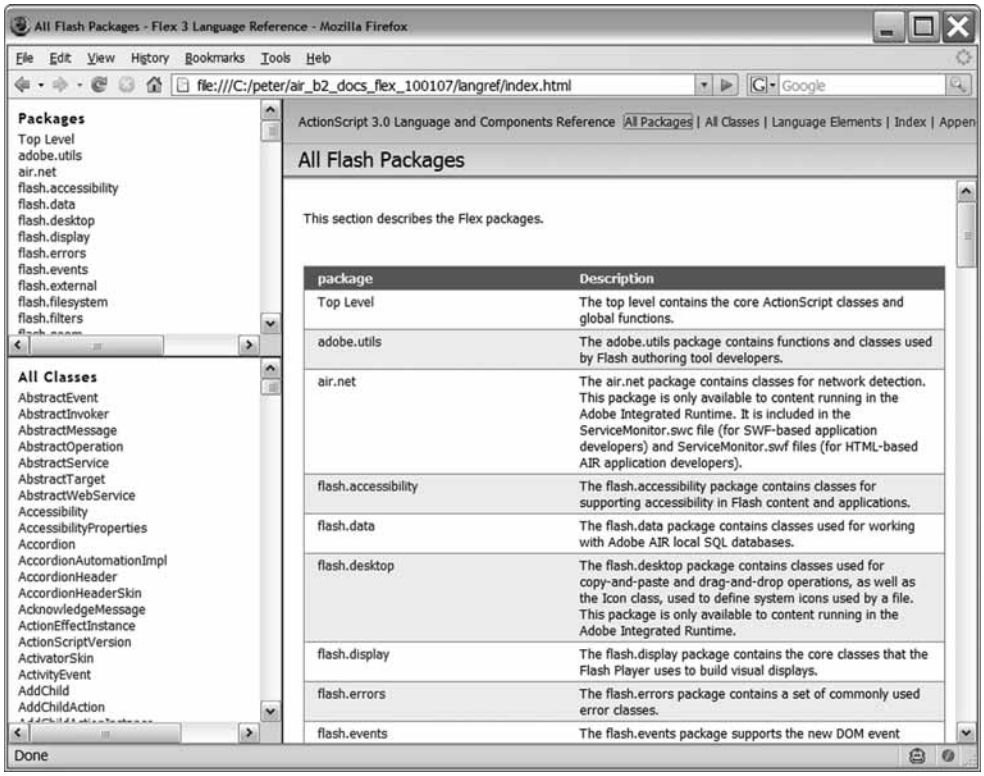

Abbildung 3.10 Die Flex 3 API-Dokumentation

3.5.5 Dummy-UI für Benutzerkonten und Anmeldung in Flex

Es ist zwar möglich, eine komplette Flex-Anwendung in einer einzigen MXML-Datei anzulegen, aber das funktioniert nur bei äußerst trivialen Anwendungen (d.h. Spielereien wie die oben gezeigte Pomodo-Anwendung mit einer einzigen Schaltfläche). Da wir in diesem Buch keine Spielzeuganwendung entwickeln, gibt es auch keinen Grund, so zu tun als ob, da wir uns damit zu einem späteren Zeitpunkt viel langweilige Arbeit mit Refactoring aufladen würden.

Also erstellen wir von Anfang an Komponeneten, die wir in ein Package namens `com.pomodo.components` legen. Da ActionScript 3 dieselbe Konvention wie Java für „Backward Domain Names“ verwendet, legen wir in app\flex ein com\pomodo\components-Verzeichnis an. Im Ordner app\flex\com\pomodo\components speichern wir unsere wiederverwendbaren MXML- und ActionScript-Komponenten.

Hinweis

ActionScript 3 unterstützt Packages mit weniger Einschränkungen als ActionScript 2 und kennt überdies auch Namensräume. Viele Details regeln, was mit Klassen, Packages und Namensräumen machbar ist und was nicht, aber wir werden es uns leicht machen und nach

dem Motto „eine Klasse pro Datei“ und „jedes Package in seinem Ordner“ vorgehen, da dies der einfachste und Java-Programmierern vertrauteste Ansatz ist.

Wir legen auch ein assets-Verzeichnis in com\pomodo\assets zum Speichern von Bildern und so weiter an. Erstellen Sie die Verzeichnisse in app\flex wie folgt:

```
c:\peter\flexiblerails\current\pomodo>cd app\flex

C:\peter\flexiblerails\current\pomodo\app\flex>mkdir com

C:\peter\flexiblerails\current\pomodo\app\flex>mkdir com\pomodo

C:\peter\flexiblerails\current\pomodo\app\flex>mkdir com\pomodo\assets

C:\peter\flexiblerails\current\pomodo\app\flex>mkdir com\pomodo\components
```

Wir erstellen zwei Komponenten: `AccountCreateBox` (hier legen Benutzer neue Konten an) und `LoginBox` (hier melden Sie sich an). Wir beginnen mit `AccountCreateBox` (siehe Listing 3.16).

Listing 3.16 app\flex\com\pomodo\components\AccountCreateBox.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"   ‖ (1)
    height="100%" label="Create Account">
    <mx:Form labelWidth="150">   ‖ (2) mx:Form: ein Layout-Tool
        <mx:FormItem required="true" label="Username">  ‖ (3) zeigt roten Stern
            <mx:TextInput id="loginTI"/>   ‖ (4)
        </mx:FormItem>
        <mx:FormItem required="true" label="Email Address">
            <mx:TextInput id="emailTI"/>
        </mx:FormItem>
        <mx:FormItem label="First Name">
            <mx:TextInput id="firstNameTI"/>
        </mx:FormItem>
        <mx:FormItem label="Last Name">
            <mx:TextInput id="lastNameTI"/>
        </mx:FormItem>
        <mx:FormItem required="true" label="Password">   ‖ (5)
            <mx:TextInput id="passwordTI"
                displayAsPassword="true"/>   ‖ (6) Zeigt Sterne statt Text
        </mx:FormItem>
        <mx:FormItem required="true" label="Confirm Password">
            <mx:TextInput id="confirmPasswordTI"
                displayAsPassword="true"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button id="createAccountButton"   ‖ (7)
                label="Create Account"/>
        </mx:FormItem>
    </mx:Form>
</mx:VBox>
```

Da die Wurzel von `AccountCreateBox` eine `VBox` **(1)** ist, ist dies auch die Klasse, von der wir eine Unterklasse bilden. Eine `VBox` ordnet ihre Kinder vertikal an, daher das V. Danach erstellen wir ein `Form` **(2)**, das diverse `FormItem`s enthält (z.B. **(3)**). Anders als die Formular-Tags in HTML sind die Tags `<mx:Form>` und `<mx:FormItem>` reine Layout-Tools.

Hinweis

Dies betone ich besonders, weil es das häufigste Missverständnis für Entwickler ist, die von HTML zu Flex kommen: Nichts geschieht, wenn die Felder in einem `mx:Form` sind; ein `Form` ist nichts weiter als ein Layout-Container – 100% Aussehen, 0% Funktionalität.

Diese `FormItem`s (z.B. **(5)**) enthalten diverse Steuerelemente, vor allem `mx:TextInput`s für Anmeldung **(4)**, E-Mail, Vor- und Nachname, Passwort **(6)** und Passwortbestätigung. Der `createAccountButton` **(7)** befindet sich ebenfalls in einem `FormItem`. Da dieses `FormItem` keine Property für die Beschriftung angibt, nimmt es als Standardwert den leeren String an. Viele `FormItem`s haben `required="true"` (z.B. **(3)**). Die einzige Auswirkung davon ist, dass neben dem Formularfeld ein roter Stern angezeigt wird.

Hinweis

Dieser rote Stern ist nur ein sichtbarer Hinweis für den Benutzer. Um das Verhalten eines obligatorischen `FormItem` hinzuzufügen, ist eine Validierung erforderlich. In Kürze kommen wir auf Validierung zu sprechen; vorläufig sollten Sie sich nur merken, dass der Wert von required keinerlei Auswirkungen auf die Kindkomponenten hat.

Achten Sie bitte auch auf die Verwendung von `displayAsPassword="true"` für `passwordTI` **(6)** und `confirmPasswordTI`. Auf diese Weise werden im `TextInput` Sterne (*) anstelle der eingegebenen Zeichen angezeigt.

Als Nächstes erstellen Sie die `LoginBox` (Listing 3.17).

Listing 3.17 app\flex\com\pomodo\components\LoginBox.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"  ‖ (1)
    height="100%" label="Login">  ‖ (2)
    <mx:Form labelWidth="150">
        <mx:FormItem required="true" label="Username">  ‖ (3)
            <mx:TextInput id="loginTI"/>
        </mx:FormItem>
        <mx:FormItem required="true" label="Password">  ‖ (4)
            <mx:TextInput id="passwordTI"
                displayAsPassword="true"/>
        </mx:FormItem>
        <mx:FormItem>  ‖ (5)
            <mx:Button id="loginButton" label="Login"/>
        </mx:FormItem>
    </mx:Form>
</mx:VBox>
```

Dies ist eine weitere `VBox` **(1)** mit einem Formular `Form` **(2)**, mit `FormItem`s, darunter ein `TextInput` für Benutzernamen **(3)**, Passwort **(4)** und ein Anmelde-Button **(5)**.

Als Nächstes holen Sie sich das Bild logo_md.png von http://www.flexiblerails.com/files/logo_md.png, indem Sie es mit der rechten Maustaste anklicken) und speichern es im Verzeichnis app\flex\com\pomodo\assets.

Den hello_controller können Sie jetzt löschen; wir sind mittlerweile über „Hello World“ hinaus. Zum Schluss bearbeiten Sie Pomodo.mxml entsprechend Listing 3.18.

Listing 3.18 app\flex\Pomodo.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:pom="com.pomodo.components.*"  ‖ (1)
    layout="vertical"
    backgroundGradientColors="[#ffffff, #c0c0c0]"
    horizontalAlign="center"  ‖ (2)
    verticalAlign="top"  ‖ (3)
    width="100%"
    height="100%">
<mx:Script>  ‖ (4)
<![CDATA[  ‖ (5)
    [Bindable]  ‖ (6)
    private var _reviews:String =  ‖ (7)
        '"pomodo, the hot new RIA by 38noises, is taking ' +
        'over Web 2.0." --Michael Arrington*\n"I wish I\'d ' +
        'invested in 38noises instead of that other company."' +
        ' --Jeff Bezos*\n"38noises closed angel funding at a ' +
        'party in my bathroom last night." --Om Malik*';
    ]]>
</mx:Script>
    <mx:Image source="com/pomodo/assets/logo_md.png" />  ‖ (8)
    <mx:Label  ‖ (9)
        text="The simple, GTD-style TODO list application."/>
    <mx:Spacer height="10"/>  ‖ (10)
    <mx:Text width="500" text="{_reviews}"/>  ‖ (11)
    <mx:Spacer height="10"/>  ‖ (12)
    <mx:Accordion width="400" height="300">  ‖ (13)
        <pom:AccountCreateBox/>  ‖ (14)
        <pom:LoginBox/>  ‖ (15)
    </mx:Accordion>
    <mx:Label text="*did not say this, but might someday!"/>  ‖ (16)
    <mx:HTTPService  ‖ (17)
        id="helloSvc"
        url="/hello/sayhello"
        method="POST"/>
    <mx:Button label="call hello service"
        click="helloSvc.send()"/>  ‖ (18)
    <mx:TextInput text="{helloSvc.lastResult}"/>  ‖ (19)
</mx:Application>
```

Als Erstes fügen wir den neuen XML-Namensraum `pom` für unsere selbst erstellten Komponenten in `com.pomodo.components` **(1)** hinzu. Danach stellen wir die horizontale und vertikale Ausrichtung gegenüber center **(2)** und top **(3)** ein. Dies wirkt sich auf die Anordnung aller Komponenten aus, die unmittelbar in der `Application` enthalten sind.

Nun binden wir erstmals **(4)** ActionScript 3-Code ein. Da das `<![CDATA[` **(5)** und `]]>` im `<mx:Script>`-Tag unerlässlich ist (damit Code anstelle von XML eingegeben werden kann), fügt Flex Builder es automatisch für Sie ein, wenn Sie `<mx:Script>` eingegeben haben. Im Moment ist nur eine Variable (`var`) namens `_reviews` **(7)** vorhanden, die einige statische Reviews als Dummy für einen Service-Aufruf enthalten, der dann Reviews anzeigen würde. Da diese Variable die Quelle einer Datenbindung ist **(11)**, müssen wir sie als `[Bindable]` kennzeichnen **(6)**. (Wäre sie eine echte Konstante und kein Dummy, würden wir `const` statt `var` verwenden und `[Bindable]` nicht verwenden, weil sich Konstanten nicht ändern.)

Jetzt fügen wir für unser schickes Logo ein `mx:Image` **(8)** hinzu, für unseren Slogan ein `mx:Label` **(9)** und dann noch einige `mx:Spacer`s **(10)(12)** rund um einen `mx:Text`, dessen `text`-Attribut **(11)** an die `_reviews`-Variable gebunden ist, die unsere gefälschten Reviews enthält **(7).** Die erfundenen Zitate von Branchengrößen sind eine schöne Motivation, die Anwendung zu erstellen, und eignen sich hervorragend als Platzhalter für die echten (und natürlich begeisterten) Kommentare, die zweifellos nur so hereinströmen werden, sobald wir auf den Markt gehen. Außerdem benötigen wir ganz im Geiste von Web 2.0 einen Namen für unser erfundenes Unternehmen und unser unechtes Produkt. Eingedenk des „me too"-Aspekts unserer Arbeit taufen wir unsere Firma 38noises (sehr komisch).

Als Nächstes erstellen wir ein `Accordion` **(13)** (damit kann man immer Punkte für schönes Aussehen sammeln), das die beiden selbst erstellten Komponenten beherbergt: `pom:AccountCreateBox` **(14)** und `pom:LoginBox` **(15)**. Beachten Sie, dass in den Komponenten (**(1)** in den früheren Listings) die `label`-Properties auf Create Account und Login eingestellt wurden – die `label`-Property wird von `Accordion` verwendet, um herauszufinden, was auf den Navigationsschaltflächen stehen soll (also die `Accordion`-Überschriften). Zum Schluss fügen wir unser Disclaimer-`Label` **(16)** hinzu und löschen den alten Code **(17)(18)(19)**.

Um das Projekt zu erstellen, suchen Sie das Pomodo-Project im Navigator und wählen aus dem Menü Project > Clean (siehe Abbildung 3.11).

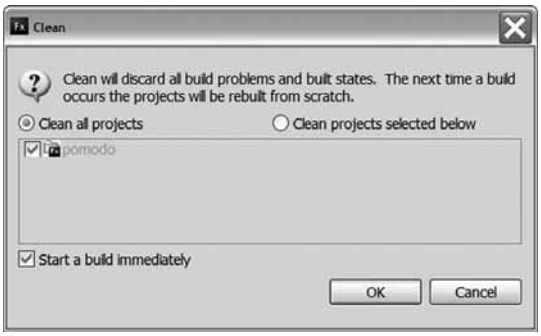

Abbildung 3.11
Eine Neuerstellung eines Projekts in Flex Builder

Belassen Sie die Markierung von „Start a Build Immediately", und klicken Sie auf OK.

Tipp

Wir müssen hier eine Neuerstellung („Clean Build") vornehmen, um sicherzustellen, dass Flex Builder das Logo findet, das wir dem Quellbaum hinzugefügt hatten. Auf diese Weise kann Flex Builder beim Build dieses Logo (und den Pfad com\pomodo\assets) in das Verzeichnis public\bin kopieren. Wenn Sie dies unterlassen, wird an Stelle des Logos vielleicht nur ein fehlendes Bild angezeigt.

Kommandozeilen-Benutzer müssen den Verzeichnisbaum com\pomodo\assets unter public\bin anlegen und die Datei logo_md.png dorthin kopieren.

Da der Build nun fertig ist, gehen Sie zu http://localhost:3000/bin/Pomodo.html (siehe Abbildung 3.12).

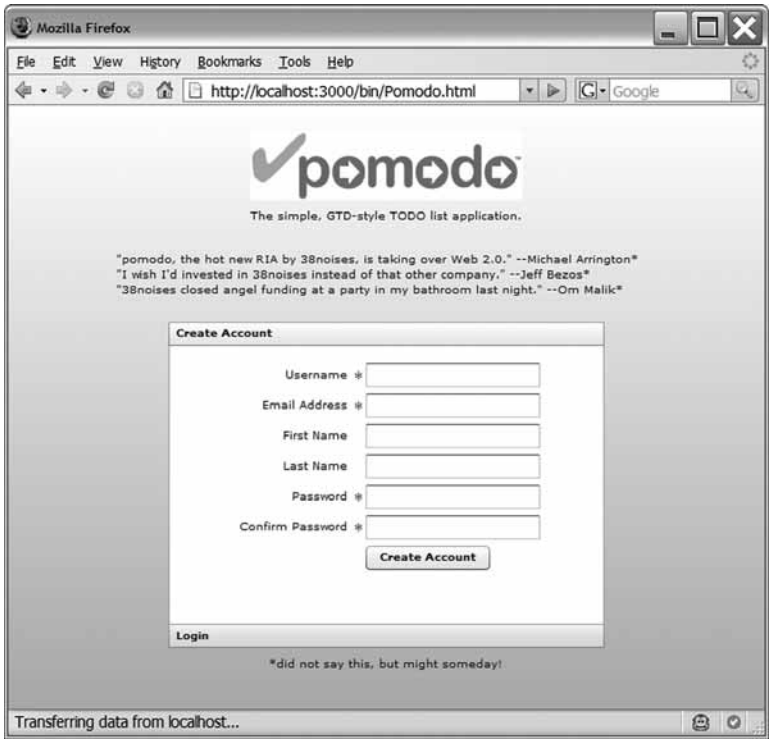

Abbildung 3.12
Der Startbildschirm

Nicht übel, so weit der Buchcode geht. Wir haben ein Web 2.0-Logo im obligatorischen Pastellgrün, ein sexy `Accordion` für das Anlegen von Benutzerkonten und die Benutzeranmeldung (wenn Sie auf den Login-Button klicken, können Sie sehen, wie das Anmeldeformular sanft ins Blickfeld rückt), einen Farbverlauf, gefälschte Zitate, die Anmeldefelder. Beachten Sie, dass die Formulare (`Form`) in der `AccountCreateBox` und der `LoginBox` ihre `labelWidth`-Properties auf 150 einstellen, damit sie schön aufgereiht erscheinen. Das sieht besser aus, wenn wir die Ansicht im `Accordion` wechseln. Es sind die Kleinigkeiten, die bewirken, dass unser Programm gefällt.

3.5.6 Funktionalität für Benutzerkonten und Anmeldung

Wenn unser Programm tatsächlich Erfolg hat, sollten wir mindestens das Anlegen von Benutzerkonten und die Benutzeranmeldung in Flex implementieren – und zwar mithilfe des `restful_authentication`-Generators. Wenn man cool ist, tut man's einfach und erhofft sich damit eine Million mehr an Geschäftswert. Das Problem ist nur, dass wir keine Ahnung haben, was REST eigentlich ist. Also lassen wir das einstweilen beiseite und überarbeiten den Code später, wenn wir mehr darüber wissen.

Als Erstes fügen wir Methoden, die mit Flex umgehen können, zu `SessionsController` und `UsersController` hinzu. Die von `SessionsController` sehen Sie in Listing 3.19.

Hinweis

Listing 3.19 zeigt Code, der in Rails so niemals geschrieben würde: Er erstellt eine zweite Methode namens `create_xml`, anstatt gleich `respond_to` zu verwenden. Das tue ich nur, weil ich REST noch nicht erklärt habe; sobald REST eingeführt wurde, werde ich natürlich ein Refactoring durchführen. Fortgeschrittene Rails-Benutzer, stört euch bitte nicht daran und lest einfach weiter.

Listing 3.19 app\controllers\sessions_controller.rb

```
# Controller für die An/Abmeldefunktion der Site.
class SessionsController < ApplicationController
  # render new.rhtml
  def new
  end

  # Wenn wir REST erklären, wird das natürlich abgeändert.
  def create_xml  ‖ (1)
    self.current_user =
      User.authenticate(params[:login], params[:password])
    if logged_in?
      if params[:remember_me] == "1"
        self.current_user.remember_me
        cookies[:auth_token] = {
         :value => self.current_user.remember_token,
         :expires => self.current_user.remember_token_expires_at
        }
      end
      render :xml => self.current_user.to_xml  ‖ (2)
    else
      render :text => "badlogin"  ‖ (3)
    end
  end

  def create  ‖ (4)
...
```

Wir erstellen im `SessionsController` eine neue Methode namens `create_xml` **(1)**, die eigentlich nur durch Kopieren, Einfügen und Verändern der `create`-Methode **(4)** zustande gekommen ist. (Wir machen uns nicht die Mühe, den kopierten Code in eine Methode umzuformen, da wir wissen, dass wir das Ganze später, wenn REST eingeführt wurde, ohnehin ein weiteres Mal einem Refactoring unterziehen müssen.) Nach erfolgreicher Anmeldung rendern wir das XML von `current_user` **(2)**, und wenn die Anmeldung scheitert, rendern wir den Text `"badlogin"` **(3)**.

Als Nächstes wird `UsersController` wie in Listing 3.20 bearbeitet.

Listing 3.20 app\controllers\users_controller.rb

```
class UsersController < ApplicationController
  # render new.rhtml
  def new
  end

  # Wenn wir REST erklären, wird das natürlich abgeändert.
  def create_xml  ‖ (1)
    @user = User.new(params[:user])
    @user.save!
    self.current_user = @user
    render :xml => @user.to_xml  ‖ (2)
  rescue ActiveRecord::RecordInvalid  ‖ (3)
```

```
      render :text => "error"
    end

    def create  ‖ (4)
...
```

Wir erstellen eine neue Methode namens `create_xml` **(1)** im `UsersController`. Diese ist ebenfalls durch Kopieren, Einfügen und Modifizieren der für uns erstellten `create`-Methode **(4)** entstanden. (Und wie oben wird auch diese nach der Umstellung auf REST durch Refactoring verschwinden.) Wird der neue Benutzer erfolgreich angelegt, rendern wir wieder das XML von `current_user` **(2)** und anderenfalls den Text `"error"` **(3)**.

Da wir nun diese beiden `create_xml`-Methoden hinzugefügt haben, wollen wir sie in Flex benutzen. Hierzu müssen wir die `AccountCreateBox` und die `LoginBox` ändern. Wir beginnen mit der `AccountCreateBox`, wie in Listing 3.21.

Listing 3.21 app\flex\com\pomodo\components\AccountCreateBox.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"
    height="100%" label="Create Account">
<mx:Metadata>                    ‖ (1) Metadaten für eigenes Event deklarieren
    [Event(name="accountCreate",
           type="com.pomodo.events.AccountCreateEvent")]
</mx:Metadata>
<mx:Script>
<![CDATA[
    import mx.controls.Alert;  ‖ (2) Import wie in Java
    import mx.rpc.events.ResultEvent;
    import com.pomodo.events.AccountCreateEvent;

    private function createAccount():void {  ‖ (3)
        svcAccountCreate.send();
    }

    private function handleAccountCreateResult(  ‖ (4)
    event:ResultEvent):void {
        var result:Object = event.result;
        if (result == "error") {
            Alert.show("Your account was not created.",
                "Error");
        } else {
            dispatchEvent(new AccountCreateEvent(XML(result)));
        }
    }
]]>
</mx:Script>
    <mx:HTTPService  ‖ (5)
        id="svcAccountCreate"
        url="/users/create_xml"
        contentType="application/xml"
        resultFormat="e4x"
        method="POST"
        result="handleAccountCreateResult(event)">
        <mx:request>
            <user>
                <login>{loginTI.text}</login>
                <email>{emailTI.text}</email>
                <first_name>{firstNameTI.text}</first_name>
                <last_name>{lastNameTI.text}</last_name>
                <password>{passwordTI.text}</password>
                <password_confirmation>
                    {confirmPasswordTI.text}
                </password_confirmation>
```

```
            </user>
        </mx:request>
    </mx:HTTPService>
    <mx:Form labelWidth="150">
        <mx:FormItem required="true" label="Username">
            <mx:TextInput id="loginTI"/>
        </mx:FormItem>
        <mx:FormItem required="true" label="Email Address">
            <mx:TextInput id="emailTI"/>
        </mx:FormItem>
        <mx:FormItem label="First Name">
            <mx:TextInput id="firstNameTI"/>
        </mx:FormItem>
        <mx:FormItem label="Last Name">
            <mx:TextInput id="lastNameTI"/>
        </mx:FormItem>
        <mx:FormItem required="true" label="Password">
            <mx:TextInput id="passwordTI"
                displayAsPassword="true"/>
        </mx:FormItem>
        <mx:FormItem required="true" label="Confirm Password">
            <mx:TextInput id="confirmPasswordTI"
                displayAsPassword="true"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button id="createAccountButton"
                label="Create Account"
                click="createAccount()"/>   ‖ (6)
        </mx:FormItem>
    </mx:Form>
</mx:VBox>
```

Wir beginnen am Ende und modifizieren den `createAccountButton`, damit er die Funktion `createAccount` **(6)** aufruft, wenn man ihn anklickt. (Sehen Sie: Wir fügen inline einen Event-Handler für ein `click`-Event hinzu, ohne weiter darüber nachzudenken – ich sagte es ja bereits: MXML wird Ihnen zur zweiten Natur!) Danach fügen wir diese `createAccount`-Funktion **(3)** in einen `<mx:Script>`-Block ein. Er ruft die `send`-Funktion eines `HTTPService` namens `svcAccountCreate` **(5)** auf, wodurch wiederum der Service mit Properties aufgerufen wird, die statisch oder durch Bindungen dafür eingestellt wurden.

Schauen wir uns die Properties von `svcAccountCreate` an. Wir setzen die `url`-Property auf `/users/create_xml`, damit ein Aufruf dieses Services die neu erstellte `create_xml`-Methode des `UsersController` in Aktion setzt. Dann stellen wir die `contentType`-Property auf `application/xml` ein, was bedeutet, dass wir den Request als XML senden. Wir stellen die `resultFormat`-Property auf `e4x` ein, um zu sagen: „Der Rückgabewert ist XML und wird wörtlich als XML in einem ActionScript-XML-Objekt zurückgegeben, auf das mit ECMAScript for XML (E4X)-Ausdrücken zugegriffen werden kann.“[5] (E4X[6] wird im Grunde genommen verwendet, um auf das Ergebnis zuzugreifen.) Als HTTP-Methode verwenden wir `POST` und als Methode zur Behandlung des Ergebnis-Events `handleAccountCreateResult`. Zum Schluss stellen wir die `request`-Property von `svcAccountCreate` auf ein XML-Dokument ein, dessen Wurzel user ist und dessen Kindknoten an die verschiedenen Steuerelemente im Formular gebunden sind.

---

[5] http://livedocs.adobe.com/flex/2/langref/mx/rpc/http/HTTPService.html#resultFormat

[6] Siehe die Flex-Dokumentation und http://life.neophi.com/danielr/2006/04/flex_2_beta_2_ actionscript_ 3_a.html – eine gute Einführung in E4X.

Hier passiert etwas ganz Subtiles: Wenn wir den Service aufrufen, stellen wir die `request`-Property ein und definieren – da diese `request`-Property XML ist – den Wert des XML als Kind des `request`-Elements. Sollte uns dieser Ansatz nicht gefallen, könnten wir auch mit `request="{someXMLVar}"` den Wert von `someXMLVar` an den Request binden.

Wenn der Service zurückkehrt, löst das Ergebnis den Event-Handler `handleAccountCreateResult` aus. Darin greifen wir auf das `event.result` zu. Wenn es den Text error enthält, zeigen wir einen Alert-Dialog mit einer kurzen Fehlermeldung. Andernfalls haben wir Erfolg gehabt und lösen daher ein neues Event namens `AccountCreateEvent` mit dem Ergebnis aus, das wir in XML umwandeln. Die Umwandlung `XML(result)` ist notwendig, weil die `user`-Variable im `Event` typisiert ist.

Hinweis

In ActionScript 3 ist XML ein nativer Typ wie `Number` und muss nicht importiert werden.

Ganz oben in der Datei deklarieren wir eigene `mx:Metadata` **(1)**, die anzeigen, dass wir ein Event senden, dessen Name `accountCreate` und dessen Typ `com.pomodo.events.AccountCreateEvent` sind. So können wir das Event einfach in der Komponente behandeln, die die `AccountCreateBox` benutzt (Sie werden gleich sehen).

Abschließend beachten Sie bitte die `import`-Anweisungen **(2)**: Diese funktionieren genau wie in Java. Die Sternsyntax (*) zum Import von Packages wird zwar ebenfalls unterstützt, aber nicht empfohlen, da sie dazu führen kann, dass überflüssige Klassen importiert werden. Das bläht die SWF-Dateien auf und verlängert die Ladezeiten. (Verwenden Sie diese Syntax nur, wenn Sie sicher sind, dass Sie jetzt und für alle Zeit jede einzelne Klasse aus einem Package benötigen.)

Da wir gerade von dem `AccountCreateEvent` reden, wollen wir es auch direkt erstellen. Legen Sie in app\flex\com\pomodo ein Event-Verzeichnis an (wenn Sie Flex Builder verwenden, können Sie mit der rechten Maustaste auf com\pomodo klicken und New > Folder wählen), und fügen Sie das neue `AccountCreateEvent` hinzu, wie in Listing 3.22 gezeigt.

Listing 3.22 app\flex\com\pomodo\events\AccountCreateEvent.as

```
package com.pomodo.events {
    import flash.events.Event;

    public class AccountCreateEvent extends Event {   ‖ (1)
        public static const ACCOUNT_CREATE:String =
            "accountCreate";   ‖ (2)

        public var user:XML;   ‖ (3)

        public function AccountCreateEvent(user:XML) {   ‖ (4)
            super(ACCOUNT_CREATE);   ‖ (5)
            this.user = user;   ‖ (6)
        }
    }
}
```

Dies ist unsere erste ActionScript 3-Klasse (na ja, nicht ganz: MXML wird in Wirklichkeit in ActionScript verwandelt; allerdings ist es die erste selbst erstellte). Sie ist darüber hinaus sehr viel einfacher: `AccountCreateEvent` erweitert `flash.events.Event` **(1)**, die Basisklasse für alle Events. Es definiert die Konstante `ACCOUNT_CREATE` **(2)** für den Namen des `Event`. Beachten Sie, dass es derselbe Name ist, den wir deklariert hatten, als wir die `Metadata` in `AccountCreateBox` sendeten. Das ist kein Zufall. Als Nächstes deklarieren wir eine öffentliche Variable `var` für den User und stellen ihren Typ auf `XML` **(3)** ein. Im Konstruktor **(4)** nehmen wir das XML für den Benutzer als Parameter entgegen. Als Erstes rufen wir den Oberklassenkonstruktor mit dem Namen des `Event` **(5)** auf und verwenden dann das empfangene XML, um die User-Instanzvariable, die wir unter **(3)** definiert haben, einzustellen **(6)**.

Nun modifizieren wir `LoginBox` (siehe Listing 3.23).

Listing 3.23 app\flex\com\pomodo\components\LoginBox.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"
    height="100%" label="Login">
<mx:Metadata>  ‖ (1)
    [Event(name="login", type="com.pomodo.events.LoginEvent")]  ‖ (2)
</mx:Metadata>
<mx:Script>
<![CDATA[
    import mx.controls.Alert;
    import mx.rpc.events.ResultEvent;
    import com.pomodo.events.LoginEvent;

    private function login():void {  ‖ (3)
        svcAccountLogin.send(  ‖ (4)
            {login: loginTI.text, password: passwordTI.text});
    }

    private function handleAccountLoginResult(  ‖ (5)
    event:ResultEvent):void {
        var result:Object = event.result;  ‖ (6)
        if (result == "badlogin") {  ‖ (7)
            Alert.show("The username or password is wrong.",  ‖ (8)
                "Login Error");
        } else {
            dispatchEvent(new LoginEvent(XML(result)));  ‖ (9)
        }
    }
]]>
</mx:Script>
    <mx:HTTPService
        id="svcAccountLogin"  ‖ (10)
        url="/sessions/create_xml"  ‖ (11)
        resultFormat="e4x"  ‖ (12)
        method="POST"  ‖ (13)
        result="handleAccountLoginResult(event)"/>  ‖ (14)
    <mx:Form labelWidth="150">
        <mx:FormItem required="true" label="Username">
            <mx:TextInput id="loginTI"/>
        </mx:FormItem>
        <mx:FormItem required="true" label="Password">
            <mx:TextInput id="passwordTI"
                displayAsPassword="true"/>
        </mx:FormItem>
```

```
            <mx:FormItem>
                <mx:Button id="loginButton" label="Login"
                    click="login()"/>  ‖ (15)
            </mx:FormItem>
        </mx:Form>
    </mx:VBox>
```

Diese Änderungen ähneln dem, was wir gerade mit `AccountCreateBox` getan haben: Als Erstes deklarieren wir `Metadata` **(1)**, die ein selbstdefiniertes `Event` **(2)** vom Typ `LoginEvent` definieren (das wir gleich erstellen werden). Als Nächstes erstellen wir eine `login` **(3)**-Funktion, die einen `svcAccountLogin HTTPService` durch Aufruf seiner `send`-Methode **(4)** in Gang setzt. Beachten Sie, dass wir die Parameter an `svcAccountLogin` als anonymes Objekt (einen Hash) übergeben **(4)**, dessen Felder innerhalb der `{}` definiert werden. Dieses anonyme Objekt funktioniert wie ein Hash in Ruby oder eine HashMap in Java als Dictionary. In ActionScript 3 können Sie anonyme Objekte auf diese Weise anlegen. Diese Praxis war in ActionScript 2 gebräuchlicher; in ActionScript 3 wird die Typisierung von Objekten angeraten. (Der Umstand, dass geschweifte Klammern auch für Bindungen verwendet werden, kann Anfänger verwirren, insbesondere, wenn Sie in diese Bindungen auch noch anonyme Objekte einsetzen!)

Tipp

Die ActionScript 3-Syntax für ein anonymes Objekt lautet `{key1: value1, key2: value2, ...}`, die Ruby-Syntax für einen Hash hingegen `{key1 => value1, key2 => value2, ...}`. Vergessen Sie nicht, in welcher Sprache Sie programmieren – ich persönlich möchte in meinen Ruby-Hashes immer Doppelpunkte verwenden, weil ich ActionScript vor Ruby erlernte!

Beachten Sie: Wenn Sie `svcAccountLogin.send()` **(4)** sagen, übersetzt Flex das anonyme Objekt in richtige HTTP-`POST`-Daten (schauen Sie nur in `log\development.log` nach, was Rails empfängt). Als Nächstes definieren wir eine Funktion namens `handleAccountLoginResult` **(5)**, die ein `ResultEvent` entgegennimmt, sich ihre `result`-Property **(6)** holt und einen `Alert` **(8)** anzeigt, wenn diese `"badlogin"` **(7)** lautet, oder im Erfolgsfalle ein benutzerdefiniertes `LoginEvent` **(9)** mit dem Ergebnis absendet. Wir definieren das `svcAccountLogin` **(10)** und setzen seinen URL **(11)** auf `"/sessions/create_xml"`, um die `create_xml`-Aktion des `SessionsController` aufzurufen. Wir geben das `resultFormat "e4x"` **(12)** und die `HTTPMethod POST` **(13)** an und fügen schließlich den Ergebnishandler hinzu **(14)**. Da wir keinen `contentType` vorgegeben haben, hat dieser standardmäßig den Typ `"application/x-www-form-urlencoded"`. Dies sind normale HTTP-`POST`-Daten, also Schlüssel-Wert-Paare. Aus diesem Grunde übergeben wir ein anonymes Objekt im `send`-Aufruf **(4)**.

Als Nächstes erstellen wir das `LoginEvent` (siehe Listing 3.24).

Listing 3.24 app\flex\com\pomodo\events\LoginEvent.as

```
package com.pomodo.events {
    import flash.events.Event;

    public class LoginEvent extends Event {  ‖ (1)
```

```
        public static const LOGIN:String = "login";  ‖ (2)

        public var user:XML;  ‖ (3)

        public function LoginEvent(user:XML) {  ‖ (4)
            super(LOGIN);  ‖ (5)
            this.user = user;  ‖ (6)
        }
    }
}
```

Dies ist eine modifizierte Version des `AccountCreateEvent`. Da die Zeilennummern zu der obigen Erklärung passen, müssen wir nicht alles noch einmal durchgehen. Nur der Name des Events ist ein anderer (`login`).

Da wir nun unsere benutzerdefinierten Events erstellt und für den Aufruf von `HTTPServices` Komponenten geschaffen haben, die mit Rails-Controllers umgehen können, müssen wir nur noch Pomodo.mxml abändern (siehe Listing 3.25).

Listing 3.25 app\flex\Pomodo.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:pom="com.pomodo.components.*"
    layout="vertical"
    backgroundGradientColors="[#ffffff, #c0c0c0]"
    horizontalAlign="center"
    verticalAlign="top"
    width="100%"
    height="100%">
<mx:Script>
<![CDATA[
    import com.pomodo.events.AccountCreateEvent;  ‖ (1)
    import com.pomodo.events.LoginEvent;

    [Bindable]
    private var _reviews:String =
        '"pomodo, the hot new RIA by 38noises, is taking ' +
        'over Web 2.0." --Michael Arrington*\n"I wish I\'d ' +
        'invested in 38noises instead of that other company."' +
        ' --Jeff Bezos*\n"38noises closed angel funding at a ' +
        'party in my bathroom last night." --Om Malik*';
    ]]>

    private function handleAccountCreate(e:AccountCreateEvent):  ‖ (2)
    void {
        showMain();
    }

    private function handleLogin(e:LoginEvent):void {  ‖ (3)
        showMain();
    }

    private function showMain():void {  ‖ (4)
        mainStack.selectedChild = mainBox;
    }
]]>
</mx:Script>
    <mx:ViewStack id="mainStack" width="100%" height="100%">  ‖ (5)
        <mx:VBox id="splashBox" horizontalAlign="center"  ‖ (6)
            verticalAlign="middle" width="100%" height="100%">
            <mx:Image source="com/pomodo/assets/logo_md.png"/>
<mx:Label text="The simple, GTD-style TODO list application."/>
```

```
            <mx:Spacer height="10"/>
            <mx:Text width="500" text="{_reviews}"/>
            <mx:Spacer height="10"/>
            <mx:Accordion width="400" height="300">
                <pom:AccountCreateBox
                    accountCreate="handleAccountCreate(event)"/>   ‖ (7)
                <pom:LoginBox login="handleLogin(event)"/>   ‖ (8)
            </mx:Accordion>
<mx:Label text="*did not say this, but might someday!"/>
        </mx:VBox>
        <pom:MainBox id="mainBox"/>   ‖ (9)
    </mx:ViewStack>
</mx:Application>
```

Als Erstes importieren wir unsere neuen Events **(1)** und erstellen anschließend Funktionen für die Behandlung von `accountCreate` **(7)** und `login` **(8)** – nämlich `handleAccount-Create` **(2)** und `handleLogin` **(3)**.

Tipp

Wenn Sie zuerst Ihre Events definieren und die `mx:Metadata`-Annotations hinzufügen, schlägt Flex Builder Ihre Events automatisch zusammen mit den Standard-Events von Flex vor. Dies ist eine gute Überprüfung, ob Sie alles richtig gemacht haben.

Beide Funktionen rufen die Funktion `showMain` **(4)** auf, die das `selectedChild` des neu erstellten `ViewStack` namens `mainStack` **(5)** auf die `mainBox` **(9)** einstellt. (Die Klasse `MainBox` werden wir gleich erstellen.) Die diversen anderen Elemente verschieben wir in eine neue `VBox` namens `splashBox` **(6)**. So bleibt alles verborgen, wenn das `mainStack` `selectedChild` wechselt.

Erstellen Sie die `MainBox`, die vorläufig nur ein Stub ist (Listing 3.26).

Listing 3.26 app\flex\com\pomodo\components\MainBox.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%" backgroundColor="#FFFFFF">
    <mx:Label text="TODO"/>
</mx:VBox>
```

Mal sehen, ob es funktioniert: Wir erstellen einen neuen Build und laden ihn neu. Der Bildschirm sieht aus wie zuvor in Abbildung 3.13. So weit, so gut. Klicken Sie auf den Login-Button, und geben Sie als Benutzernamen `ludwig` und als Passwort `foooo` ein.

Klicken Sie auf den Login-Button. Der Bildschirm sollte wie in Abbildung 3.14 aussehen.

Hurra!

Nun wollen wir das Anlegen von Benutzerkonten testen. Gehen Sie zu http://localhost:3000/bin/Pomodo.html, und erstellen Sie einen neuen Benutzer mit dem Namen `peter`, der E-Mail-Adresse `peter@pomodo.com`, dem Vornamen `Peter` und dem Nachnamen `Armstrong` sowie dem Passwort `foooo` (Abbildung 3.15).

Wenn Sie auf Create Account klicken, sollte derselbe TODO-Bildschirm wie in Abbildung 3.12 beim Anmelden erscheinen.

Abbildung 3.13 Anmeldeformular ausfüllen

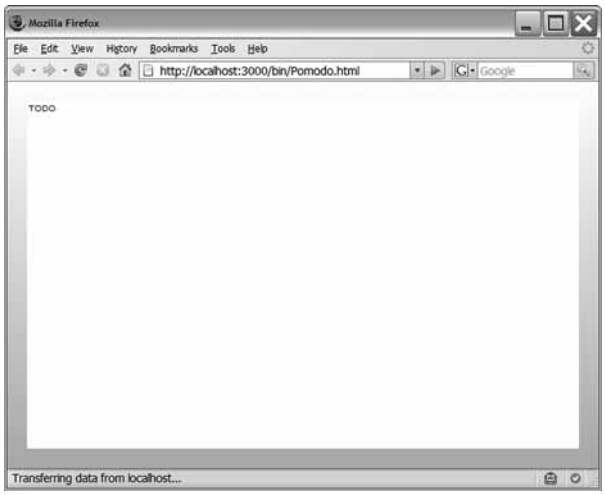

Abbildung 3.14 Angemeldet

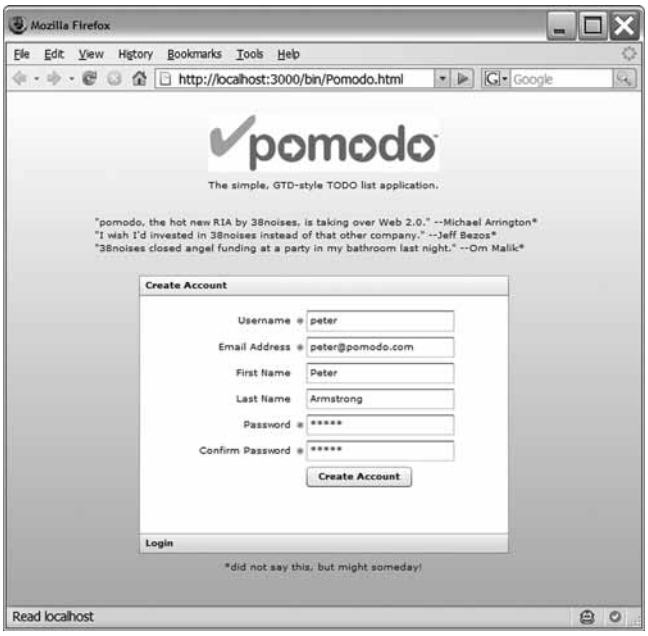

Abbildung 3.15 Ein Konto anlegen

Bevor wir in Begeisterung ausbrechen, müssen wir allerdings noch einige lose Enden miteinander verknüpfen. Erstens zeigt der Bildschirm in Abbildung 3.12 eine `MainBox`, die sich nicht über die gesamte Breite des Browsers erstreckt: Wir sehen immer noch etwa zehn Pixel von unserem Farbverlauf. Nun mag ich zwar Farbverläufe sehr gerne, aber be-

vor wir das zu einem Feature erklären, müssen wir eingestehen, dass wir es eigentlich nicht wollten. Also müssen wir diesen Fehler beheben. Und da wir gerade dabei sind, können wir gleich ein kleines Refactoring beginnen und die `splashBox` in eine eigene, benutzerdefinierte Komponente verlagern sowie einige kleine Layout-Änderungen vornehmen (siehe Listing 3.27).

Listing 3.27 app\flex\com\pomodo\components\SplashBox.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:pom="com.pomodo.components.*"
    horizontalAlign="center" verticalAlign="top"
    width="100%" height="100%">
<mx:Metadata>  ‖ (1)
    [Event(name="accountCreate",  ‖ (2)
           type="com.pomodo.events.AccountCreateEvent")]
    [Event(name="login", type="com.pomodo.events.LoginEvent")]  ‖ (3)
</mx:Metadata>
<mx:Script>
<![CDATA[
    import com.pomodo.events.AccountCreateEvent;
    import com.pomodo.events.LoginEvent;

    [Bindable]
    private var _reviews:String =
        '"pomodo, the hot new RIA by 38noises, is taking ' +
        'over Web 2.0." --Michael Arrington*\n"I wish I\'d ' +
        'invested in 38noises instead of that other company."' +
        ' --Jeff Bezos*\n"38noises closed angel funding at a ' +
        'party in my bathroom last night." --Om Malik*';
]]>
</mx:Script>
    <mx:VBox width="500" horizontalAlign="center">
        <mx:Image source="com/pomodo/assets/logo_md.png" />
        <mx:Label
           text="The simple, GTD-style TODO list application."/>
        <mx:Spacer height="10"/>
        <mx:Text width="100%" text="{_reviews}"/>
        <mx:Spacer height="10"/>
        <mx:Accordion width="400" height="300">
            <pom:AccountCreateBox/>  ‖ (4)
            <pom:LoginBox/>  ‖ (5)
        </mx:Accordion>
        <mx:Label text="*did not say this, but might someday!"/>
    </mx:VBox>
</mx:VBox>
```

Die ganze Datei ist zwar neu, stammt aber im Wesentlichen aus Pomodo.mxml. Beachten Sie, dass wir mit `Metadata` **(1)** auch deklarieren, dass wir die Events `accountCreate` **(2)** und `login` **(3)** senden. Außerdem sollten Sie zur Kenntnis nehmen, dass wir diese nicht auf `pom:AccountCreateBox` **(4)** und `pom:LoginBox` **(5)** behandeln. Wir hätten das zwar tun können, hätten aber in diesem Fall ein anderes Event senden müssen, um es von Pomodo.mxml behandeln zu lassen. Würden wir ein Event wie `showMain` erstellen, verlören wir allerdings Informationen (nämlich darüber, was geschehen ist: ein neues Konto oder eine Benutzeranmeldung). Dann könnten wir nicht mehr so gut selbst definierte Aktionen als Antwort auf diese spezifischen Events anstoßen.

Nun modifizieren wir erneut Pomodo.mxml. Eine Durchsicht der API-Docs fördert zu Tage, dass die Lücke wahrscheinlich durch die Werte von `paddingLeft` usw. entstanden ist. Diese Werte ändern wir und löschen den Code, den wir in die `SplashBox` verschoben haben (Listing 3.28).

Listing 3.28 app\flex\Pomodo.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:pom="com.pomodo.components.*"
    layout="vertical"
    backgroundGradientColors="[#ffffff, #c0c0c0]"
    horizontalAlign="center"
    verticalAlign="top"
    paddingLeft="0"  ‖ (1)
    paddingRight="0"
    paddingTop="0"
    paddingBottom="0"
    width="100%"
    height="100%">
<mx:Script>
<![CDATA[
    import com.pomodo.events.AccountCreateEvent;
    import com.pomodo.events.LoginEvent;

    [Bindable]  ‖ (2)
    private var _reviews:String =
        '"pomodo, the hot new RIA by 38noises, is taking ' +
        'over Web 2.0." --Michael Arrington*\n"I wish I\'d ' +
        'invested in 38noises instead of that other company."' +
        ' --Jeff Bezos*\n"38noises closed angel funding at a ' +
        'party in my bathroom last night." --Om Malik*';

    private function handleAccountCreate(e:AccountCreateEvent):
    void {
        showMain();
    }

    private function handleLogin(e:LoginEvent):void {
        showMain();
    }

    private function showMain():void {
        mainStack.selectedChild = mainBox;
    }
]]>
</mx:Script>
    <mx:ViewStack id="mainStack" width="100%" height="100%">
        <mx:VBox id="splashBox" horizontalAlign="center"  ‖ (3)
            verticalAlign="middle" width="100%" height="100%">
...
            </mx:Accordion>
<mx:Label text="*did not say this, but might someday!"/>
        </mx:VBox>
        <pom:SplashBox id="splashBox"  ‖ (4)
            accountCreate="handleAccountCreate(event)"
            login="handleLogin(event)"/>
        <pom:MainBox id="mainBox"/>
    </mx:ViewStack>
</mx:Application>
```

Als Erstes stellen wir die `padding`-Attribute ein **(1)**, anschließend löschen wir die `_reviews` **(2)** und sämtlichen Code in der `splashBox` **(3)** und ersetzen dies durch unsere neue `SplashBox` **(4)**. Beachten Sie, dass wir die `accountCreate`- und `login`-Events in der `SplashBox` behandeln, wobei wir dieselben Methoden aufrufen, die auch zur Event-Behandlung in der `AccountCreateBox` und der `LoginBox` herangezogen wurden.

Nach einem neuen Build und Reload gehen Sie jetzt zum Anmeldeformular und geben wieder den Benutzernamen `ludwig` und das Passwort `foooo` ein.

Nichts geschieht.

Der Grund dafür ist simpel: Wir hatten uns darauf verlassen, dass die Events von der `AccountCreateBox` oder `LoginBox` auf wundersame Weise durch die `SplashBox` gehen. Doch wie sich zeigt, ist das zwar möglich, aber nicht ohne die Events zu modifizieren. (Swing-Entwickler: Ratet mal, wie die Property heißt, die wir einstellen müssen.)

Die hierzu notwendige `Event`-Property heißt `bubbles`, und sie muss `true` sein, damit das Event in der Komponentenhierarchie nach oben „sprudeln“ kann. Als ehemaliger Java-UI-Entwickler nehme ich an, dass dies eine Leistungsoptimierung ist. Die `bubbles`-Property ist das zweite Argument des `Event`-Konstruktors, und wir werden sie jetzt benutzen (Listing 3.29 **(1)** und Listing 3.30 **(2)**).

Listing 3.29 app\flex\com\pomodo\events\AccountCreateEvent.as

```
package com.pomodo.events {
    import flash.events.Event;

    public class AccountCreateEvent extends Event {
        public static const ACCOUNT_CREATE:String =
            "accountCreate";

        public var user:XML;

        public function AccountCreateEvent(user:XML) {
            super(ACCOUNT_CREATE, true);  ‖ (1)
            this.user = user;
        }
    }
}
```

Listing 3.30 app\flex\com\pomodo\events\LoginEvent.as

```
package com.pomodo.events {
    import flash.events.Event;

    public class LoginEvent extends Event {
        public static const LOGIN:String = "login";

        public var user:XML;

        public function LoginEvent(user:XML) {
            super(LOGIN, true);  ‖ (2)
            this.user = user;
        }
    }
}
```

Wenn Sie jetzt einen neuen Build erstellen, das Projekt neu laden und sich als ludwig anmelden, erscheint der Bildschirm aus Abbildung 3.16.

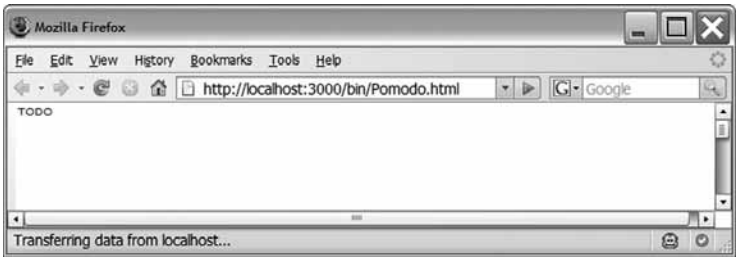

Abbildung 3.16
Der Stub von `MainBox`

Das `login`-Event wurde erhört und die `MainBox` angezeigt. Die Änderung der `padding`-Attribute hat sich ausgezahlt: Vom Farbverlauf ist nichts mehr zu sehen.

Nun laden Sie die Anwendung neu und versuchen, sich mit gefälschten Anmeldedaten einzuloggen. Es müsste etwas wie in Abbildung 3.17 passieren.

Abbildung 3.17 Falsche Anmeldedaten

Beachten Sie, wie die Anwendung ausgegraut und deaktiviert hinter dem modalen, leicht transparenten Alert-Dialog liegt. Wer hätte gedacht, dass ein Anmeldefehler so sexy aussehen könnte?

Da wir nun Benutzer anlegen und anmelden können, sollten wir als Nächstes Beispieldaten hinzufügen, die automatisch für uns angelegt werden, wenn wir newdb.bat ausführen. So müssen wir den Benutzer ludwig nicht jedesmal neu erstellen, wenn wir beschließen, von vorne anzufangen. (Da wir später noch mehr Daten hinzufügen, können wir von jeder be-

liebigen Iteration aus starten und haben immer eine funktionierende, mit Daten ausgestattete Datenbank zur Verfügung, anstatt mühsam zu überlegen, welche Daten wir als Nächstes laden müssen.)

3.6 Den Test-Fixtures Daten hinzufügen

Wir könnten die Daten mithilfe eines SQL-Skripts hinzufügen (was in früheren Versionen dieses Buchs auch wirklich getan wurde), wollen uns aber von keinem bestimmten Datenbanksystem abhängig machen. Die natürliche Wahl wäre daher, Datenmigrationen zu verwenden (siehe auch AWDwR, S. 274), um die Testdaten zu laden. Es gilt jedoch als schlechter Stil, auf diese Weise irgendetwas anderes als Referenzdaten zu laden.

Hinweis

Im Abschnitt „Data Migrations“ (S. 276) in *AWDwR* sehen Sie ein Beispiel für diese Ermahnung: „Seien Sie gewarnt: Die einzigen Daten, die Sie in Migrationen laden sollten, sind Daten, die Sie auch in der Produktion heranziehen würden: Lookup-Tabellen, vordefinierte Benutzer und dergleichen. Laden Sie keine Testdaten auf diesem Wege in Ihre Anwendung.“

Was tun? Müssen wir denn unbedingt ein SQL-Skript verwenden?

Zum Glück nicht – Fixtures[4] kommen uns zur Hilfe!

Hinweis

Ehrlich gesagt bin ich nicht sicher, ob der nachfolgend beschriebene Ansatz auch für eine normale Entwicklungsaufgabe der richtige Weg wäre. Aber um ein Lehrbuchbeispiel (insbesondere in diesem Buch) iterativ zu entwickeln, funktioniert er gut. Deshalb verwende ich ihn in diesem Buch, und nicht etwa, weil ich ihn als „Best Practice“ bezeichnen würde.

Wenn wir den Authentifizierungsgenerator ausführen, erstellt er eine Testfixture namens users.yml in test/fixtures. Zu dieser gehören zwei Benutzer (quentin und aaron), und wir fügen zwei weitere hinzu (ludwig und wolfgang); siehe Listing 3.31.

Listing 3.31 test\fixtures\users.yml

```
quentin:
  id: 1
  login: quentin
  email: quentin@example.com
  salt: 7e3041ebc2fc05a40c60028e2c4901a81035d3cd
  crypted_password: 00742970dc9e6319f8019fd54864d3ea740f04b1 # test
  created_at: <%= 5.days.ago.to_s :db %>


aaron:
  id: 2
  login: aaron
  email: aaron@example.com
  salt: 7e3041ebc2fc05a40c60028e2c4901a81035d3cd
```

[4] Fixture = Haltevorrichtung *(d. Übers.)*

```
  crypted_password: 00742970dc9e6319f8019fd54864d3ea740f04b1 # test
  created_at: <%= 1.days.ago.to_s :db %>


ludwig:  ‖ (1)
  id: 3
  login: ludwig
  email: lvb@pomodo.com
  first_name: Ludwig
  last_name: van Beethoven
  salt: cf1bc466e9dedd7d687e967ba37947971d44ab6e
  crypted_password: fc3f2237b0edbeab2c08eecbc7bd6ef9b2124080 # foooo
  created_at: <%= 5.days.ago.to_s :db %>


wolfgang:  ‖ (2)
  id: 4
  login: wolfgang
  email: wam@pomodo.com
  first_name: Wolfgang
  last_name: Mozart
  salt: 945da846ec9a4f29c10e21789bfb213d62f8fee5
  crypted_password: 2f6ffa90ee8a87c2121b813eb68265130f9b1410 # barrr
  created_at: <%= 1.days.ago.to_s :db %>
```

Die Benutzer ludwig **(1)** und wolfgang **(2)** basieren auf quentin und aaron, fügen jedoch den `first_name` und `last_name` hinzu und haben die Kennungen 3 und 4 (weil diese die nächsten in der Reihenfolge sind). Die Werte von `salt` und `crypted_password` werden auf das eingestellt, was `mysql` geliefert hat, als wir die Benutzer ludwig und wolfgang manuell anlegten. (Na ja, vielleicht nicht Sie, aber ich habe sie vor diesem Schritt angelegt.)

Nun modifizieren wir die Datei newdb.bat, um die Testfixtures **(1)** nach Ausführung der Migrationen zu laden (Listing 3.32).

Listing 3.32 newdb.bat

```
mysql -h localhost -u root -p <db\create.sql
call rake db:migrate
call rake db:fixtures:load  ‖ (1)
```

Halten Sie den Server an, führen Sie newdb.bat aus, und starten Sie Ihren Server erneut:

```
C:\peter\flexiblerails\current\pomodo>newdb.bat

C:\peter\flexiblerails\current\pomodo>
mysql -h localhost -u root -p  0<db\create.sql
Enter password: *******

C:\peter\flexiblerails\current\pomodo>call rake db:migrate
c:0:Warning: require_gem is obsolete.  Use gem instead.
(in C:/peter/flexiblerails/current/pomodo)
== 1 CreateUsers: migrating ===================================================
-- create_table("users", {:force=>true})
   -> 0.1090s
== 1 CreateUsers: migrated (0.1090s) ==========================================

c:0:Warning: require_gem is obsolete.  Use gem instead.
(in C:/peter/flexiblerails/current/pomodo)

C:\peter\flexiblerails\current\pomodo>ruby script\server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
```

Schauen Sie nun in mysql nach:

```
mysql> select id, login, email, first_name, last_name from users;
+----+----------+---------------------+------------+---------------+
| id | login    | email               | first_name | last_name     |
+----+----------+---------------------+------------+---------------+
|  1 | quentin  | quentin@example.com | NULL       | NULL          |
|  2 | aaron    | aaron@example.com   | NULL       | NULL          |
|  3 | ludwig   | lvb@pomodo.com      | Ludwig     | van Beethoven |
|  4 | wolfgang | wam@pomodo.com      | Wolfgang   | Mozart        |
+----+----------+---------------------+------------+---------------+
4 rows in set (0.00 sec)

mysql>
```

Melden Sie sich abschließend als ludwig an (oder als wolfgang, quentin oder aaron), um sich zu vergewissern, dass alles immer noch funktioniert.

Wenn Sie denken, es sei eher unwahrscheinlich, dass Testdaten aus den Fixtures in die Entwicklungsdatenbank geladen werden, sollten Sie wissen, dass wir tatsächlich Ad-hoc-Tests durchführen werden, wenn wir zur Entwicklungszeit mit der Anwendung herumspielen. Ganz abwegig ist dies also nicht.

Bevor wir diese Iteration beenden, führen wir einen Test durch, um uns zu vergewissern, dass wir keine größeren Schäden angerichtet haben.

3.7 Die Tests überprüfen

Wir müssen unsere Tests überprüfen. Wenn beispielsweise ein Test gezählt hat, wie viele Benutzer von den Fixtures erstellt wurden, hätten wir diesen Test jetzt zerstört, weil wir neue Benutzer hinzugefügt haben. (Der restful_authentication-Generator hat einige Tests für uns produziert; siehe test\functional und test\unit.) Diese wollen wir nun mit dem `rake`-Befehl ausführen (Standardziel ist die Ausführung der Tests, also dasselbe wie `rake test`. Sagen Sie `rake --tasks`, um alle verschiedenen Tasks anzuzeigen, aus denen Sie auswählen können):

```
c:\peter\flexiblerails\current\pomodo>rake
12345678901234567890123456789012345678901234567890123456789012345678901234
c:0:Warning: require_gem is obsolete.  Use gem instead.
(in c:/peter/flexiblerails/current/pomodo)
c:/ruby/bin/ruby -Ilib;test "c:/ruby/lib/ruby/gems/1.8/gems/
     ➥rake-0.7.3/lib/rake/rake_test_loader.rb"
     ➥"test/unit/user_test.rb"
Loaded suite c:/ruby/lib/ruby/gems/1.8/gems/rake-0.7.3/lib/rake/rake_test_loader
Started
.............
Finished in 0.953 seconds.

13 tests, 26 assertions, 0 failures, 0 errors
c:/ruby/bin/ruby -Ilib;test "c:/ruby/lib/ruby/gems/1.8/gems/
     ➥rake-0.7.3/lib/rake/rake_test_loader.rb"
     ➥"test/functional/sessions_controller_test.rb"
     ➥"test/functional/users_controller_test.rb"
Loaded suite
     ➥c:/ruby/lib/ruby/gems/1.8/gems/rake-0.7.3/lib/rake/rake_test_loader
Started
```

```
..............
Finished in 0.875 seconds.

14 tests, 26 assertions, 0 failures, 0 errors
c:/ruby/bin/ruby -Ilib;test "c:/ruby/lib/ruby/gems/1.8/gems/
     ➥rake-0.7.3/lib/rake/rake_test_loader.rb"

c:\peter\flexiblerails\current\pomodo>
```

Die 13 Tests in unit\user_test.rb funktionieren ebenso wie die 14 funktionalen Tests in test\functional. Wir dürfen also hoffen, dass wir nicht alles zerstört haben und dass wir diese nun abgeschlossene Iteration als Grundlage für Größeres und Besseres verwenden können.

Vor dem Ende dieser Iteration möchte ich Ihnen zwei weitere Dinge ans Herz legen: Konfigurieren Sie Flex Builder, und richten Sie Subversion und Subclipse ein. Subversion wird in Anhang A behandelt, und wie Sie Flex Builder konfigurieren können, um pomodo auszuführen und dezubuggen, wird im folgenden Teilabschnitt beschrieben. Wenn Sie Flex Builder nicht benutzen, können Sie gleich zum nächsten Teilabschnitt springen.

Subversion und Subclipse

In der echten Entwicklungsarbeit ist die Wahrscheinlichkeit groß, dass Sie Subversion benutzen werden, es sei denn, Sie sind einer von diesen coolen Typen, die zu Git (http://git.or.cz/) gewechselt sind. Wie wird Subversion für Flex und Rails eingerichtet? Ich könnte einfach sagen: „Sorgen Sie dafür, dass es das Verzeichnis public\bin ignoriert“, aber da diese Frage häufiger gestellt wird, gebe ich Ihnen im Anhang A einige zusätzliche Hinweise. Wenn Sie Subversion mit Flex und Rails verwenden möchten und eine Kurzeinführung wünschen, wird Ihnen dieser Anhang helfen.

3.8 Flex Builder für Ausführung und Debugging von pomodo konfigurieren

Es wäre doch nett, wenn wir einfach den kleinen grünen Kreis mit dem „Play“-Dreieck darin anklicken könnten, um pomodo auszuführen. Doch was noch wichtiger ist: Flex Builder 3 enthält einen richtigen Debugger, dessen Verwendung wir erlernen müssen. Beide Aufgaben packen wir nun an.

Als Erstes klicken Sie auf den grünen play-Button, den Sie in Abbildung 3.18 sehen.

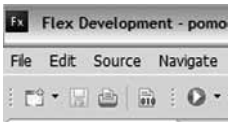

Abbildung 3.18
Der Run-Button unter dem Navigate-Menü

Das setzt die Anwendung zwar in Gang, aber der URL wird Ihre Entsprechung von file:///C:/peter/flexiblerails/current/pomodo/public/bin/Pomodo.html sein. Das ist nicht, was wir wollten: Wir möchten http://localhost:3000/bin/Pomodo.html laden, damit Rails

die Seite mit unserer Flex-Anwendung serviert. Also müssen wir das Ziel der Programmausführung konfigurieren. Klicken Sie auf den Pfeil neben dem Run-Button, und wählen Sie Other, wie in Abbildung 3.19 gezeigt.

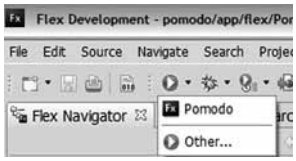

Abbildung 3.19
Run-Ziele durch Auswahl von „Other“ konfigurieren

Wir sehen den Run-Dialog. Da wir das Pomodo-Projekt gerade durch den Run-Button ausgeführt haben, befindet sich im Flex Application-Ordner bereits eine Startkonfiguration namens Pomodo – und diese ist es, die die falschen URLs hat. Wir müssen nur die URLs in http://localhost:3000/bin/Pomodo.html abändern, und fertig. Wie in Abbildung 3.20 gezeigt, deaktivieren Sie Use Defaults und bearbeiten die URLs.

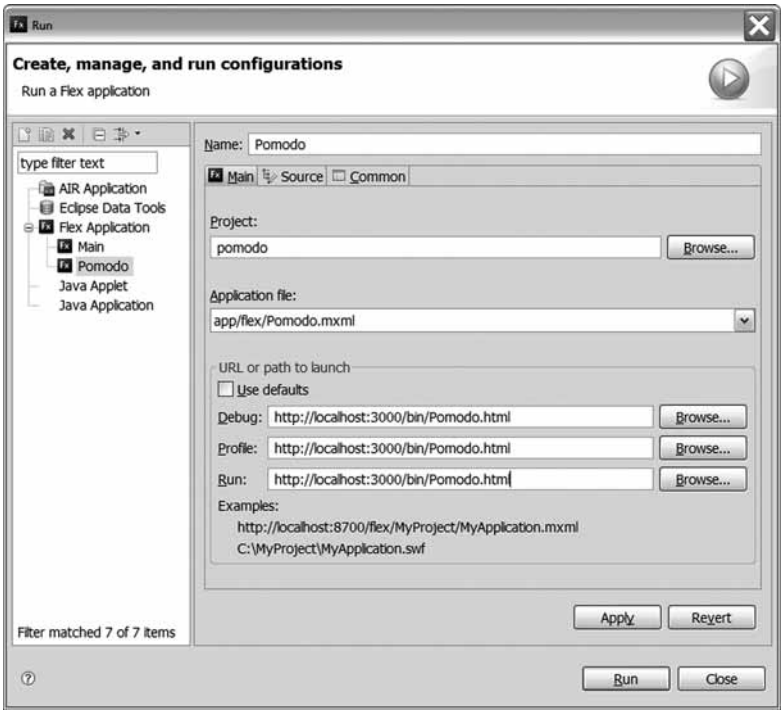

Abbildung 3.20 Reparatur der Pomodo-URLs im Flex Application-Abschnitt des *Run*-Konfigurationsdialogs

Hinweis

Flex SDK-Benutzer: Es gibt auch den Kommandozeilen-Debugger fdb.
Siehe http://www.adobe.com/devnet/flex/articles/client_debug_08.html.

Speichern Sie die Einstellung durch einen Klick auf den Apply-Button, und klicken Sie dann auf Close, um den Dialog zu schließen. Nun können wir mit einem Klick auf die Symbolleistenschaltfläche Run die pomodo-Anwendung ausführen und sie mit einem Klick auf Debug debuggen.

3.9 Zusammenfassung

In dieser Iteration haben wir viel geleistet. Wir haben die Benutzeranmeldung in Rails funktionsfähig gemacht und die Flex-Benutzeroberfläche daran angeschlossen. Dabei erlernten wir die Grundlagen von Routing in Rails und Flex. Außerdem ist dies ein guter Ausgangspunkt für jede Flex+Rails-Anwendung, natürlich besonders für pomodo (die allerdings noch Mängel aufweist, wie Iteration 5 zeigen wird).

Die nächste Iteration ist nicht nur viel kürzer, sondern auch richtig spaßig. Wir sammeln darin die Anforderungen und gestalten und erstellen fast die gesamte Flex-UI – mit viel weniger Code, als Sie vermuten.

▶ Der Code bis zu diesem Punkt liegt im Ordner iteration03.